

Reconocimiento de voz por redes neuronales aplicado a un robot LEGO.

Trabajo conjunto de los cursos de doctorado de:

- Redes neuronales
- Robótica Autónoma

UNED 2003

Víctor Padilla Martín-Caro

vpadilla@inicia.es

Tfno :918935277

Asignaturas Doctorado (2002/2003)

- Redes Neuronales
- Robótica Autónoma

1.- Introducción.

Uno de los retos mas complejos y apasionantes al que nos podemos enfrenta en el mundo científico es sin duda el de intentar imitar el comportamiento del ser humano por medio de aparatos y sistemas avanzados. En este sentido, resulta complicado conseguir máquinas que caminen con naturalidad, que posean articulaciones semejantes a nuestras manos, que sean capaces de entender el habla, etc. En el presente trabajo nos hemos planteado un doble objetivo. Por un lado adentrarnos en el mundo de las redes neuronales para crear un sistema que sea capaz de reconocer o, al menos, discriminar con una fiabilidad aceptablemente alta, entre una serie de palabras o sílabas pertenecientes a notas musicales. Por el otro, la creación de un robot que se comuniquen en tiempo real con el ordenador y sea capaz de ejecutar las órdenes provenientes de éste (actuadores), teniendo en cuenta el entorno y la posición en la que se encuentra (sensores).

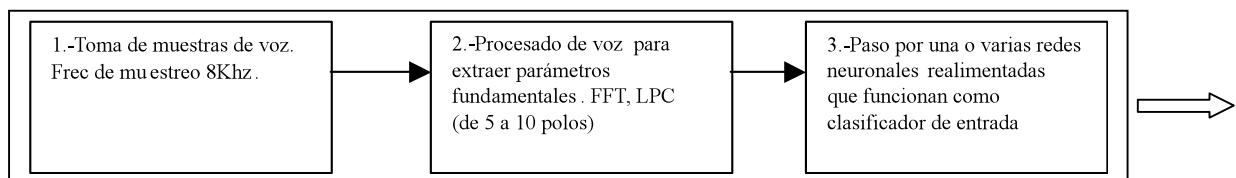
El presente trabajo pretende ser un estudio introductorio a un trabajo mas profundo, posible tesis doctoral, sobre reconocimiento de voz en redes neuronales y sus posibles aplicaciones prácticas en robótica.

2.- Resumen del trabajo

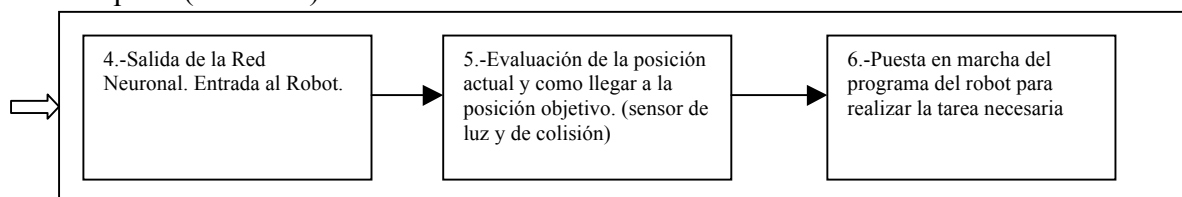
El trabajo consiste en “fabricar” un robot que distinga entre 5 palabras monosílabas, correspondientes a 5 notas musicales (do, re ,mi, fa, sol) y sea capaz de tocarlas en el teclado de un piano estándar.

Paso a indicar el esquema por medio de un gráfico.

Bloque 1 (Redes)



Bloque 2 (Robótica)



Como podemos observar, nos enfrentamos a una serie de problemas multidisciplinares, que implican aspectos de:

- Informática (programación C++, Vbasic...)
- Tratamiento de señal
- Fonética (acústica)
- Redes neuronales (algoritmo back-propagation)
- Mecánica
- Robótica

A continuación explicamos brevemente cada uno de los bloques anteriores.

2.1.-Muestreo y Conversión A/D: Las señales de voz son convertidas en señales eléctricas por medio de un micrófono, que posteriormente, serán transformadas de analógico a digital. Para la palabra,

el rango de frecuencias donde se encuentra la mayor parte de la información y la energía está entre 1000Hz y 3000Hz. [Recuero 83] Por tanto, estableciendo la frecuencia de muestreo en 8000Hz, sabemos por los teoremas de Shannon y Nyquist, [Shannon 49] que podemos reproducir perfectamente señales de hasta 4000Hz, suficientes para nuestro propósito.

2.2.- Procesado de voz: De nuestra señal de voz, podremos extraer una información mas interesante si pasamos del dominio del tiempo al de la frecuencia. Para esto emplearemos el teorema de Fourier, en concreto, la transformada discreta de fourier(DFT) facilitado por el algoritmo FFT [Jackson 89]. Un conocido problema es la distorsión producida al procesar directamente X muestras de voz en el FFT. Para evitar esto, empleamos una función ventana (bien Hamming o Hanning). Cuando la ventana de análisis aumenta, la frecuencia de resolución aumenta, puesto que el tiempo de resolución disminuye. Por otro lado, cuando la longitud de la ventana de análisis se acorta, el tiempo de resolución aumenta y la frecuencia de resolución disminuye.

Los parámetros LPC [Furui 89] son una técnica muy importante de estimación espectral, porque pueden proporcionar una indicación de los polos (y de los formantes) de la función de transferencia del tracto vocal. El mayor inconveniente es que al calcular sólo los polos, la extracción de LPC funciona bien en sonidos sonoros, pero no es demasiado fiable en sonidos sordos. Aún así, lo vamos a utilizar, porque extraer entre 5 y 10 polos es una buena aproximación de la envolvente de la FFT y una reducción considerable en el número de parámetros y, por tanto, en el número de nodos de nuestra red.

2.3.- ANN a implementar: Basándonos en los trabajos previos de David P. Morgan y C. L. Scofield [Morgan 91] vamos a implementar una red multicapa realimentada, con la siguiente estructura:

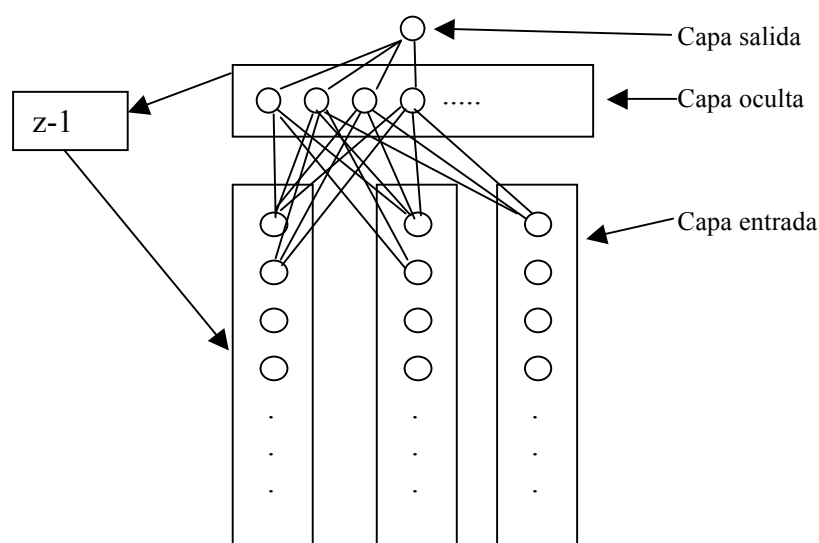


Fig.1

Vamos a incluir parámetros LPC en la capa de entrada en lugar de un array de filtros resonantes fijos a una determinada frecuencia. En nuestro caso, tendremos 5 redes entrenadas, una para cada palabra. La salida mas probable, siempre superando un umbral, gana.

La implementación de la red será en C++ creando unas dll como librerías de funciones que puedan ser importadas en un programa con una interfaz mas sencilla de programar, como Visual Basic. De esta manera intentamos obtener la rapidez del C para el procesado de señal y la facilidad de programación de Visual Basic para el entorno del programa.

2.4.-Entrada al robot: Para la parte de robótica, vamos a emplear el kit de Lego RIS 1.5 [Bagnan 02], empleando las comunicaciones infrarrojas que soporta, intercambiando información por medio del lenguaje VisualBasic 6.0 de Microsoft, a través del ocx proporcionado por Lego. Dentro del RCX introduciremos un programa que detecte tanto el valor de los sensores, como los valores de los parámetros de salida proporcionados por la red.

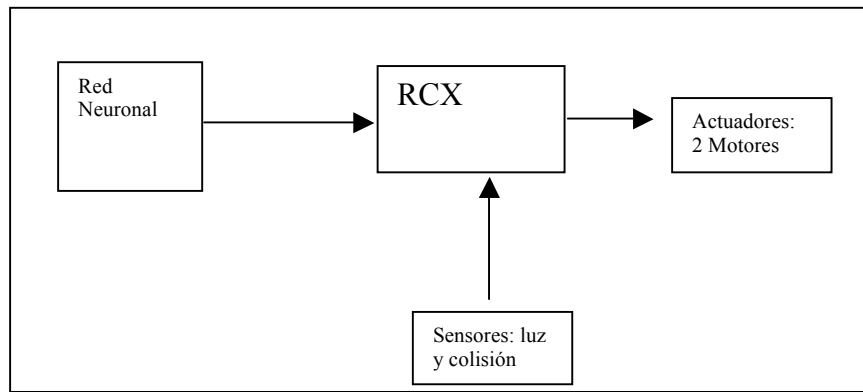


Fig.2

2.5.-Estructura del Robot: El robot empleado es el siguiente.

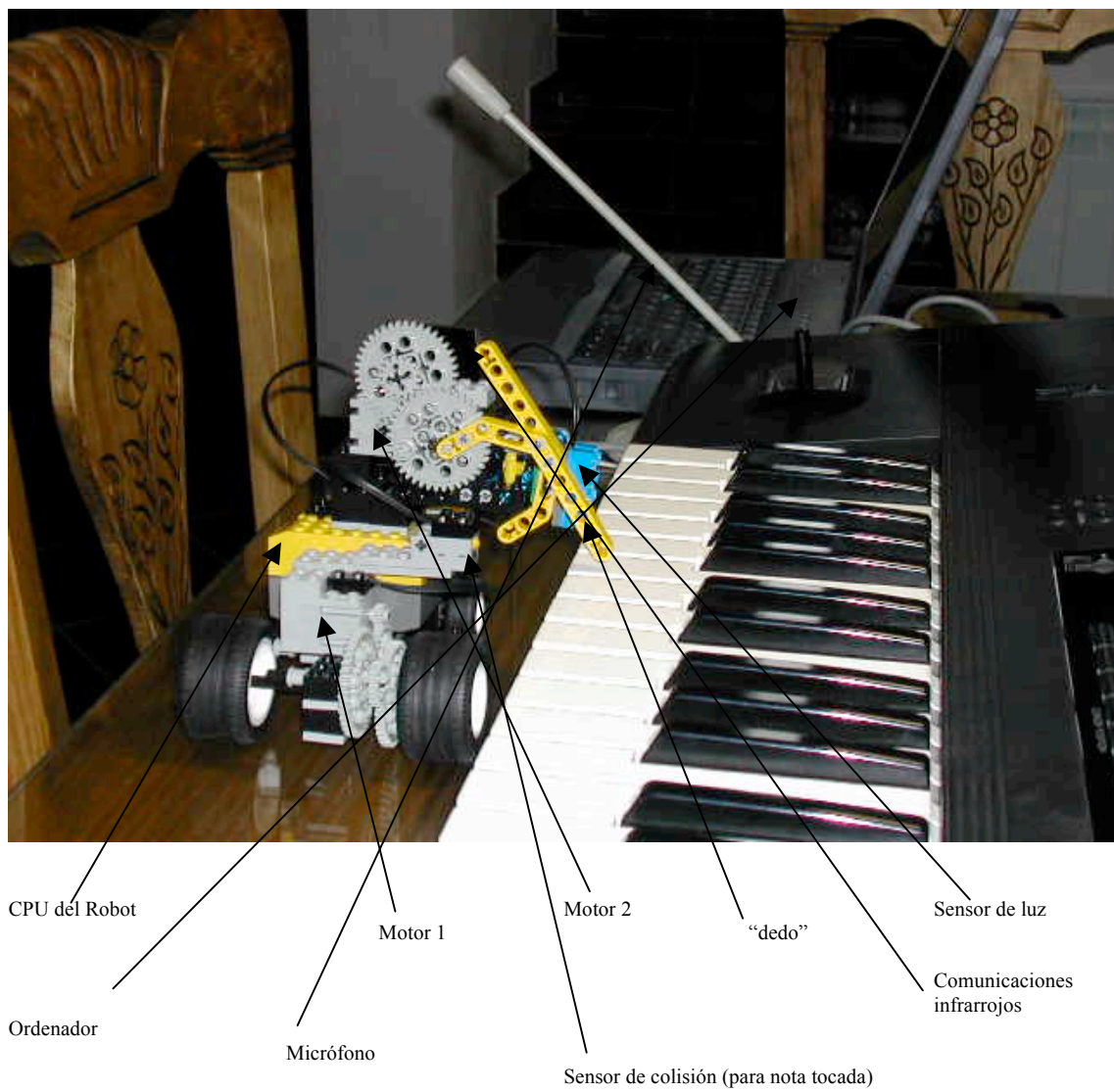


Fig.3

2.6.- Algoritmo básico del Robot.

El esquema básico puede ser el siguiente:

- Enviamos el programa compilado al RCX
- Ajuste de sensor de luz (en función de la luz de la sala y del blanco de la tecla)
- Si evento de nueva nota
 - o Calculamos la posición actual de nota
 - o Buscamos cómo llegar a la nueva nota (sensor de luz)
 - Puesta en marcha de motor 1
 - Contamos los pasos de tecla
 - Si no detecta pasos de tecla en un tiempo T, error.
 - Si hay paso de tecla, el siguiente no puede estar a una distancia ni superior a X ni inferior a Y
 - o Llegada a la nota deseada. Stop motor 1
 - o Pulsamos tecla (motor 2) hasta activar sensor de presión
 - o Levantamos “dedo” (motor 2 sentido inverso un tiempo T)
 - o Listos para recibir el nuevo evento de nota.

3.-Análisis y estudios previos a la implementación del proyecto.

3.1 Análisis espectral de la voz.

La voz se analiza normalmente por medio de sus características espectrales tales como el espectro de frecuencia o la función de autocorrelación, en lugar de usar directamente la forma de onda. Hay dos importantes razones para esto. Una es que la señal de voz es reproducible sumando ondas sinusoidales y la amplitud y fase de estas ondas cambia lentamente. La otra, la forma de percibir la señal de voz por el oído humano, dando principal importancia al contenido espectral y no a su fase. La densidad de potencia espectral puede ser considerado como el producto de dos elementos: la envolvente espectral, que cambia lentamente como una función de la frecuencia y la estructura espectral fina, que cambia rápidamente. La estructura espectral fina produce patrones periódicos para sonidos sonoros, pero no para sonidos sordos.

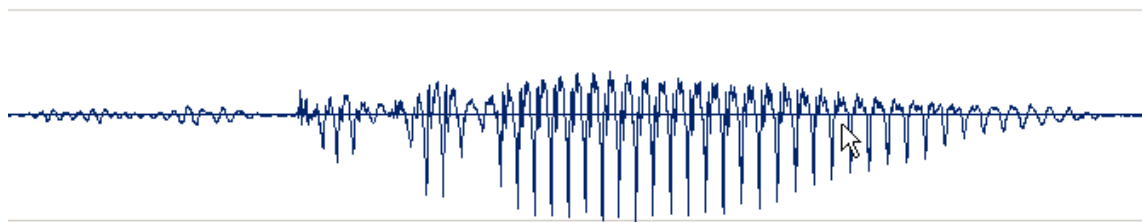


Fig. 4

La figura 4 muestra la sílaba Re en tiempo muestreada a 8kHz. El número de muestras es de 2800. En la figura 5, esa misma señal está pasada al dominio de la frecuencia (0-4000Hz). Puede observarse un contenido espectral fino extraído por medio de la FFT y una envolvente que varía lentamente (LPC). En este caso hemos empleado 10 polos para la envolvente del algoritmo LPC.

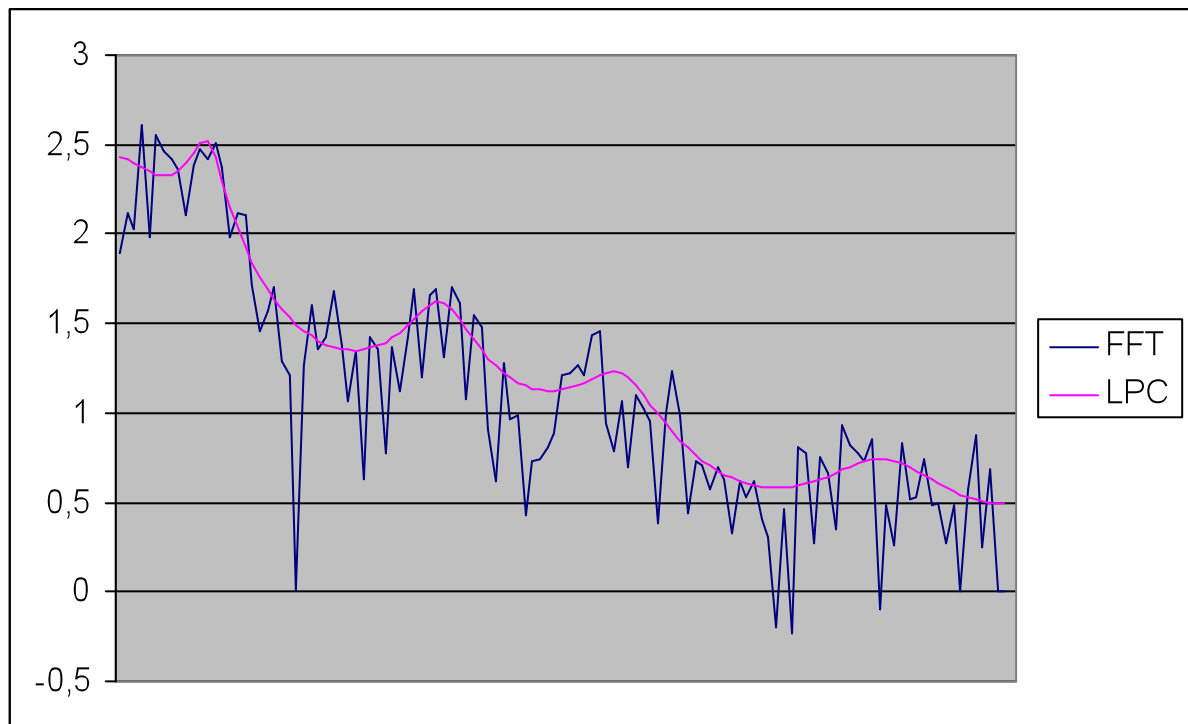


fig.5

El algoritmo para la transformada de Fourier ha sido el siguiente:

$$S(\omega) = \frac{1}{2\pi N} \left| \sum_{n=0}^{N-1} x(n) e^{-j\omega n} \right|^2$$

donde:

$$\omega = 2\pi fT \quad (f=\text{frecuencia real})$$

N =número de muestras.

Implementación

La implementación del algoritmo en C++ está en el fichero FFT.CPP

La definición de la función en C++ es la siguiente:

```
float(*fft(float *X, int N))
```

donde X representa un puntero a un array de datos X de dimensión N. Un ejemplo de uso sería:

```
float *ffts;
ffts=fft(X,240);
```

3.2.- Función Ventana.

Con el fin de extraer parámetros de la señal de voz, ésta debe ser multiplicada por una apropiada ventana en el dominio del tiempo. Esta multiplicación tiene dos efectos. Primero, atenúa gradualmente la amplitud de los bordes del intervalo de extracción para evitar un cambio abrupto. Segundo, produce la convolución en frecuencia de la función ventana y la señal de voz. Es por tanto deseable que la función ventana satisfaga dos características para reducir la distorsión espectral. Una es una alta resolución en frecuencia, con un estrecho y agudo lóbulo principal. La otra es una baja pérdida espectral del resto de las frecuencias producida por la convolución.

Puesto que estas dos condiciones son contradictorias y es imposible satisfacer ambas, existen una serie de funciones que llegan a un compromiso. Entre éstas, la ventana Hamming se define como:

$$W_H(n) = 0.54 - 0.46 \cos\left(\frac{2n\pi}{N-1}\right)$$

Esta ventana es normalmente empleada en análisis de voz. La ventana hamming tiene la ventaja de que su resolución en el dominio de la frecuencia es relativamente alta y su atenuación espectral es pequeña, puesto que la atenuación del primer lóbulo lateral es mayor de 43 db. [Jackson 89]

Otra ventana empleada es la ventana Hanning

$$W_N(n) = 0.5 - 0.5 \cos\left(\frac{2n\pi}{N-1}\right)$$

Aunque la ventaja de esta ventana es que sus lóbulos laterales de orden alto son mas pequeños que los de la ventana Hamming, la atenuación del primer lóbulo lateral es únicamente de 30 dB.

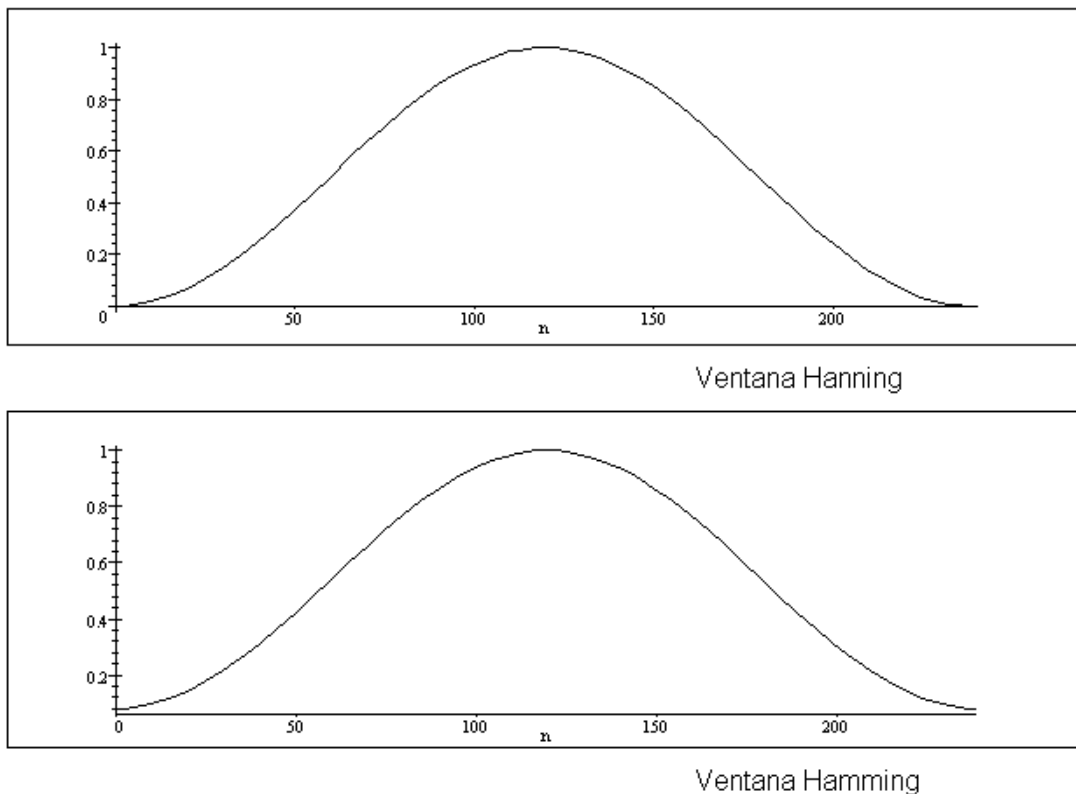


Fig. 6

En la figura 6 hemos representado estas dos ventanas en el dominio del tiempo.

Implementación

Para nuestro trabajo hemos empleado la ventana Hamming por los motivos expuestos anteriormente. La implementación de la ventana está en el fichero HAMMING.CPP

La definición de la función en C++ es la siguiente:

```
float *hamming(int N)
```

Esta función calcula la función ventana Hamming devolviendo la dirección de un array de datos de dimensión N.

```

Ejemplo: float datos[240], dat_ttf[240], *ham, *ffts;
          Ham=hamming(240);
          For(x=0;x<=239;x++)
          {
            dat_fft[x]=datos[x]*ham[x];
          }
          ffts=fft(dat_fft, 240); //Calcula la FFT de datos con
                                   //ventana Hamming de 240

```

la relación entre el periodo de muestreo $T[s]$, el número de muestras a analizar N y la resolución en frecuencia Δf se expresa como:

$$\Delta f = \frac{1}{TN}$$

Se esta formula deducimos que la resolución se incrementa proporcionalmente al intervalo de voz a analizar. Por ejemplo, en nuestro caso:

$T=0.125\text{ms}$ (8kHz de muestreo)
 $N=240$ muestras (30 ms de duración)

$$\Delta f = \frac{8000}{240} = 33.3\text{Hz}$$

Cuando la ventana de análisis aumenta, la frecuencia de resolución aumenta puesto que el tiempo de resolución disminuye. Además, cuando la forma de onda es multiplicada por una ventana, bien Hamming o Hanning, la longitud del intervalo efectivo de análisis disminuye aproximadamente en un 40%. Por tanto, el intervalo de análisis debe ser solapado a lo largo de la señal de voz para facilitar el seguimiento de las variaciones del espectro con el tiempo.

El intervalo de análisis multiplicado por una ventana se denomina trama. En este trabajo vamos a realizar un solape de 80 muestras con ventanas Hamming de 240 muestras. Gráficamente se puede representar de la siguiente manera:

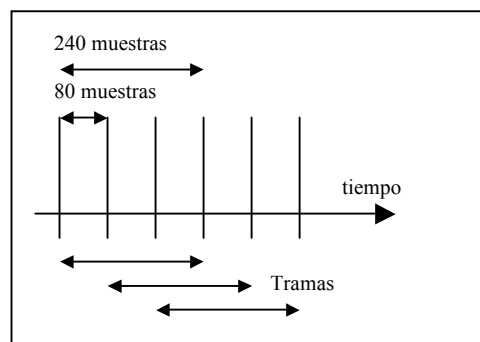


Fig. 7

3.3.- LPC (Linear predictive Coding)

Los parámetros LPC son una técnica muy importante de estimación espectral porque pueden proporcionar una indicación de los polos (y de los formantes) de la función de transferencia del tracto vocal [Shaughnessy 90]. El algoritmo LPC intenta predecir el valor de cualquier punto en un sistema lineal variante en el tiempo, basado en los valores de las P muestras anteriores. La representación, solamente de polos de la función de transferencia del tracto vocal, $H(z)$, se puede representar por la siguiente ecuación:

$$H(z) = \frac{G}{1 - \sum_{i=1}^P a(i)z^{-i}}$$

Los valores de $a(i)$ se denominan coeficientes de predicción, mientras que G representa la amplitud o ganancia, asociada con la excitación del tracto vocal. La notación z^{-1} indica un retardo en tiempo en el dominio de las transformadas en Z .

Los polos de la función de transferencia de la ecuación anterior se determinan de las raíces del polinomio del denominador. Debido a que el modelo LPC es un modelo “todo polos”, éste puede capturar las frecuencias de resonancia, o formantes, pero no los ceros, que son importantes para los sonidos nasales. Además, LPC no estima adecuadamente las señales que no tienen polos, tal como los sonidos sordos y el ruido. De hecho, componentes no lineales tales como el ruido, afectan de forma adversa a la estimación LPC.

El Modelo LPC

Para la señal de voz $s(n)$ producida por un sistema lineal, la muestra de voz predicha $\bar{s}(n)$ es una función de $a(i)$ y de las anteriores muestras de voz, de acuerdo con:

$$\bar{s}(n) = \sum_{i=1}^P a(i)s(n-i)$$

El análisis LPC pretende obtener los términos $a(i)$ de acuerdo con el criterio del mínimo error cuadrático. El error se define como:

$$e(n) = s(n) - \bar{s}(n) = s(n) - \sum_{i=1}^P a(i)s(n-i)$$

Tomando la derivada del error cuadrático medio con respecto a los coeficientes $a(i)$ e igualándola a 0 resulta:

$$\frac{\partial}{\partial a(j)} \left[s(n) - \sum_{i=1}^P a(i)s(n-i) \right]^2 = 0$$

$$\left[s(n) - \sum_{i=1}^P a(i)s(n-i) \right] s(n-j) = 0 \quad \text{para } 1 \leq j \leq P$$

Por tanto,

$$s(n)s(n-j) = \sum_{i=1}^P a(i)s(n-i)s(n-j) \quad \text{para } 1 \leq j \leq P$$

Hay varios métodos para resolver esta ecuación de coeficientes de predicción. Comentaremos el método de autocorrelación con el algoritmo de Levinson-Durbin [Furui 89], por ser éste el algoritmo implementado para extraer los parámetros LPC en nuestro proyecto.

Método de Autocorrelación (Levinson-Durbin)

Este método supone que la señal es estacionaria dentro de la ventana de análisis. Esta suposición se puede satisfacer con una función ventana $w(m)$ ($0 \leq m \leq N-1$) donde N corresponde a un número de discretos puntos en un intervalo de tiempo fijo. La solución de autocorrelación a la ecuación anterior se puede expresar como:

$$R(j) = \sum_{i=1}^P a(i)R(i-j) \text{ para } 1 \leq j \leq P$$

donde $R(j)$ es una función par ($R(j)=R(-j)$) y se calcula de la siguiente ecuación:

$$R(j) = \sum_{m=0}^{N-1-j} s(m)s(m+j) \text{ para } 0 \leq j \leq P$$

Una vez que el término de la autocorrelación $R(j)$ ha sido calculado, usamos un algoritmo recursivo, denominado recursión de Durbin para determinar el valor de $a(i)$. El estado inicial de la recursión comienza con un término de energía.

$$E^0 = R(0) \quad G^2 = E^0$$

Para calcular los siguientes estados realizamos las siguientes ecuaciones:

$$k(i) = (R(i) - \sum_{j=1}^{i-1} a^{i-1}(j)R(i-j)) / E^{i-1} \text{ para } 1 \leq i \leq P$$

$$a^i(i) = k(i)$$

$$a^i(j) = a^{(i-1)}(j) - k(i)a^{i-1}(i-j) \text{ para } 1 \leq j \leq i-1$$

$$E^i = (1 - k(i)^2)E^{i-1}$$

La solución final de $a(j)$ es dada por $a^P(j)$ para $1 \leq j \leq P$

Implementación

La implementación de este algoritmo en C++ se encuentra en el fichero LPC.CPP

La definición en C++ es la siguiente:

*float(*lpc(float *X, int polos))*

Esta función calcula los parámetros LPC de un array de datos, indicando el array y el número de polos a extraer. La función devuelve un puntero a la dirección del primer valor en que se encuentran los valores de LPC. El primer valor $a[0]$ indica la energía, y el resto ($a[1]...a[\text{polos}]$), son los valores de la función LPC mediante el método “Levinson-Durbin”.

Ejemplo: `float X1[240], *a;`
`a=lpc(X1,10); //Calcula 10 parámetros LPC de X1`

3.4.- Algoritmo “back-propagation”

El algoritmo “back-propagation” es el empleado en el entrenamiento de las redes del tipo perceptrón multicapa de 3 o más capas. Si la red da una respuesta incorrecta, se retropropaga el error y se corrigen los pesos de las conexiones para disminuir ese error. La topología general de una red de estas características es la representada en la figura 8.

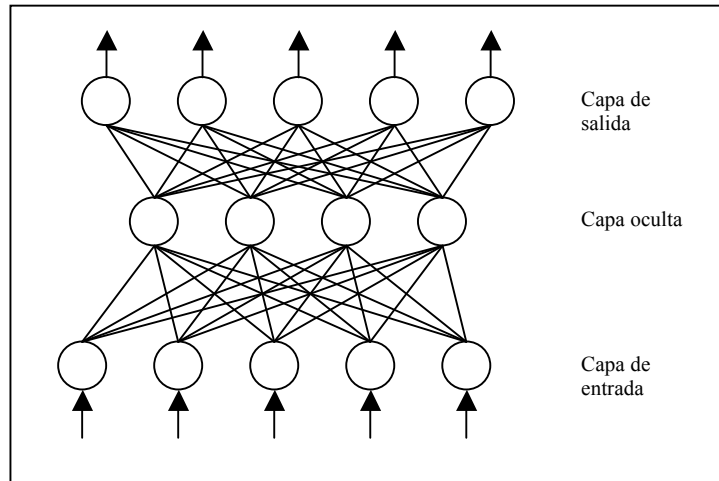


Fig.8

Las capas están completamente interconectadas, cada unidad de proceso está conectada a todas las unidades de la capa anterior y posterior, aunque no conectadas a las de la misma capa. En la figura 9 mostramos una unidad de proceso.

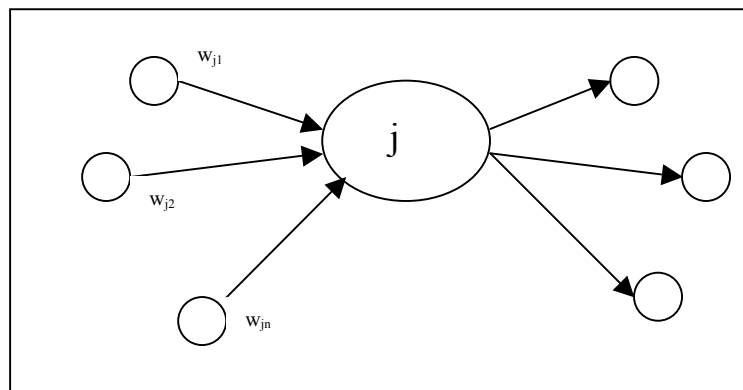


Fig. 9

La unidad de proceso tiene asociados el valor de la suma de las entradas (S_j), un valor de salida (a_j) y un valor de error (δ_j) que se usa durante el ajuste de los pesos.

Forward-propagation

Este algoritmo comienza cuando un patrón se presenta a la entrada de la red. Después de la activación de la primera capa, se produce una propagación al resto de ellas hasta determinar el valor de salida. Los valores de salida llegan a la unidad j y son sumadas mediante la siguiente fórmula:

$$S_j = \sum_i a_i w_{ji}$$

donde:

a_i =nivel de activación de la unidad i
 w_{ji} =peso desde la unidad i hasta la unidad j

Después de calcular la suma, ésta pasa a través de la función sigmoide. La principal ventaja de esta función es que es derivable y nos permitirá retropropagar el error en capas ocultas, como veremos posteriormente. Su formula es:

$$f(x) = \frac{1}{1 + e^{-x}}$$

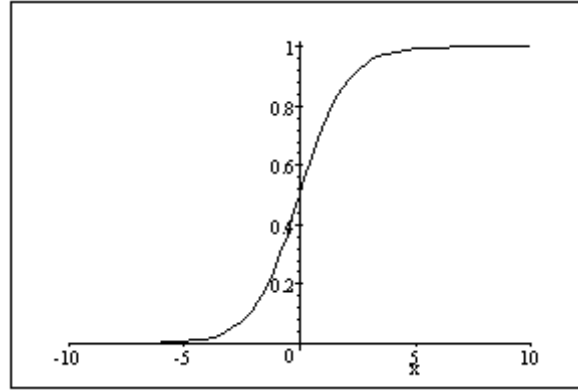


Fig. 10

Backward Propagation

Cada unidad de proceso en la capa de salida proporciona un único número real que es comparado con la salida deseada. Basados en esta diferencia, calculamos un valor de error para cada unidad en la capa de salida. Con estos valores de error podemos calcular el resto de valores que ajusten los pesos de la red. El error se denota por la variable δ y para la capa de salida se calcula de la siguiente manera.

$$\delta_j = (t_j - a_j) f'(S_j)$$

donde

t_j =valor deseado u objetivo (target)

a_j =valor de salida de la unidad j

$f'(x)$ = derivada de la función sigmoide

S_j = suma de los pesos de las entradas a la unidad j

La cantidad $(t_j - a_j)$ refleja la cantidad de error. El término f' hace que el factor de corrección sea mayor o menor dependiendo del punto de la curva en el que nos encontremos. Si $f(x)$ es la función sigmoide, se puede demostrar que:

$$f'(x) = f(x)(1 - f(x))$$

lo cual facilita notablemente los cálculos.

Para calcular el error en una capa oculta, empleamos la siguiente expresión:

$$\delta_j = \left[\sum_k \delta_k w_{kj} \right] f'(S_j)$$

Para ajustar los pesos de las conexiones empleamos los valores de δ obtenidos. Este ajuste se realiza teniendo en cuenta la expresión siguiente:

$$\Delta w_{ji} = \eta \delta_j a_i$$

La variable η en la ecuación de ajuste de pesos es la relación de aprendizaje. Su valor (normalmente entre 0.25 y 0.75) se escoge por el usuario. Valores muy grandes pueden conducir a inestabilidad de la red, valores pequeños a entrenamientos lentos. Esta variable también puede variar a lo largo del entrenamiento.

La principal ventaja es este tipo de redes radica en su tremenda posibilidad de aprender patrones diferentes. Esta red no requiere ningún conocimiento a priori de las funciones que van de la entrada a la salida, por lo que pueden ser usadas en un amplio espectro de posibilidades, en nuestro caso, para tareas de reconocimiento de voz, como veremos posteriormente. El principal inconveniente es el tiempo de entrenamiento necesario hasta minimizar el error. Pueden ser necesarias cientos e incluso miles de iteraciones para conseguir la convergencia de la red. Otro problema es la posibilidad de llegar a mínimos locales en el entrenamiento con lo que la red puede quedar “atrapada” en un mínimo local y no llegar al entrenamiento óptimo. Existen diversos algoritmos para intentar solventar este problema.

Redes Recurrentes

Las redes neuronales recurrentes se caracterizan por tener dos caminos: Un camino directo hacia la salida y un lazo de realimentación. El motivo principal de usar redes recurrentes para modelos temporales es que permite representar el efecto que tiene el tiempo en la secuencia a procesar. En la mayoría de las redes recurrentes, la activación de los nodos en la capa q en el instante t se almacena en unos nodos de contexto (context node) que pueden ser usados como entrada (generalmente dentro de la misma capa) en el instante t+1. Los pesos entre los nodos de contexto y los nodos a los que alimentan se denominan pesos de contexto (context weight). La figura siguiente ilustra una red recurrente completamente interconectada con nodos de contexto. Los nodos de contexto los representamos con cuadrados. Este es el esquema de implementación que usaremos en el presente trabajo.

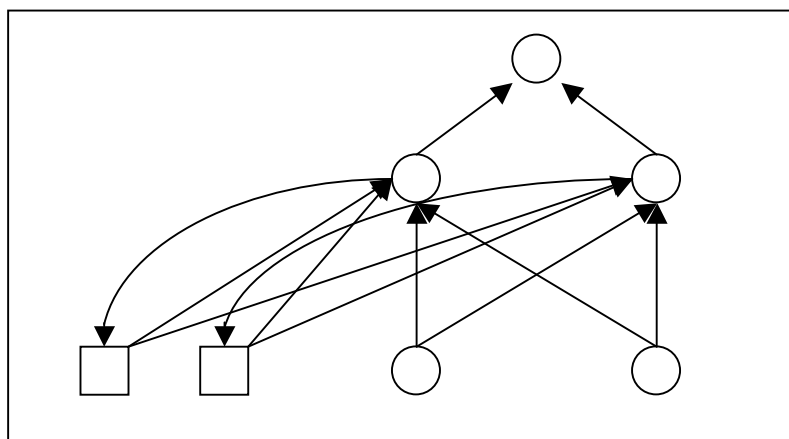


Fig.11

Modelo de Flujo Temporal

El modelo de flujo temporal diseñado por Watrous y Shastri [Watrous 87] usa una red de tres capas en la cual cada nodo de la capa oculta y de la capa de salida contiene autotransiciones con un tiempo de retardo. La red de flujo temporal permite a los pesos de contexto que sean actualizados por “back-propagation”. Con el fin de entrenar esta red, la salida deseada puede ser expresada como una función del tiempo. Esto es un cambio significativo con respecto a las redes estáticas en las cuales la respuesta deseada es fija para un determinado patrón. La selección de una apropiada función respuesta dependiente del tiempo es un proceso empírico; sin embargo, la especificación del comportamiento de la red en tiempo es muy importante porque la función objetivo (target) determina como serán actualizados los pesos de la red.

Una típica función objetivo puede ser la siguiente:

$$r(t) = 0.5(1 + t/T) \quad \text{tipo rampa}$$

$$r(t) = 1 - \alpha e^{-\beta t/T} \quad \text{tipo exponencial}$$

donde:

T=tiempo total

t=cada una de las subdivisiones de T

Si lo que queremos es que se produzca un rechazo a la entrada, la función será:

$$1 - r(t)$$

Implementación

Todo el algoritmo back-propagation de la red recurrente con el entrenamiento siguiendo el modelo del flujo temporal, en C++, se encuentra en el fichero `entrenam2003.cpp` optimizado para el caso particular de la red implementada. Por medio del proyecto **entrenamDLL** en C++ generamos un fichero DLL con una función con la que comunicarnos con el proyecto global en Vbasic. En esta DLL hacemos referencia a las funciones de los ficheros `hamming.cpp` y `lpc.cpp`, comentadas con anterioridad.

Para hacer uso de esta DLL en Vbasic tenemos que hacer una llamada a la función:

```
Private Declare Function EntrenamRNN Lib "entrenam.dll" (ByVal FicheroEntrada As String, ByVal Iteraciones As Integer, ByVal FicheroSalida As String) As Boolean
```

```
'Llamada a la función. Retorna false si error  
boolEntrenam = EntrenamRNN(txtEntrada,intIteraciones, txtSalida)
```

txtEntrada: path del fichero con los datos del entrenamiento.

IntIteraciones: número de iteraciones en el entrenamiento

txtSalida: path del fichero donde se escribe la red entrenada

La implementación se encuentra en el fichero `frmEntrenamiento.frm` del proyecto `UNED_VB_2003` de Visual Basic.

4.-Implementación del robot.

El robot fue realizado mediante el kit de lego mindstorms 1.5. La comunicación con el robot la realizamos en Vbasic mediante el ocx spirit.ocx incluido en el CD de instalación. Uno de los principales problemas fue el limitado número de sensores y actuadores incluidos en el kit. Como solamente disponemos de dos motores, empleamos uno para el movimiento del robot y otro para el brazo encargado de tocar la nota. El sensor de luz (de bastante poca calidad) lo empleamos para distinguir los pasos de nota y el sensor de contacto para determinar cuando el dedo del robot ha finalizado de tocar la nota. El esquema es el siguiente:

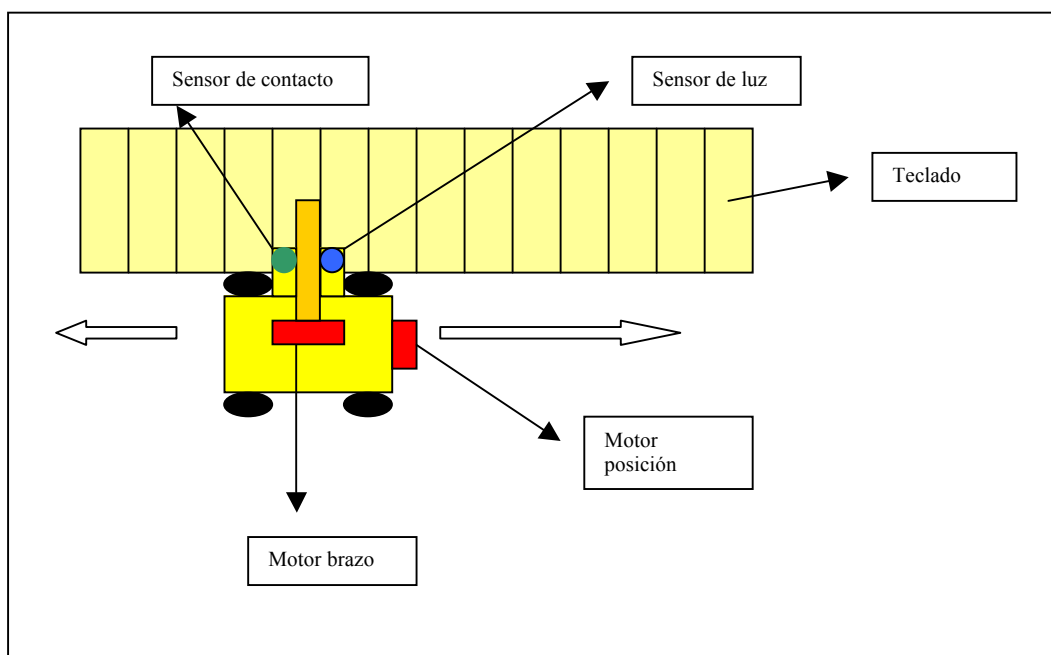


Fig. 12

La programación del robot y las pruebas de funcionamiento se encuentran en el formulario frmLego y el módulo modLego del proyecto UNED_VB_2003.

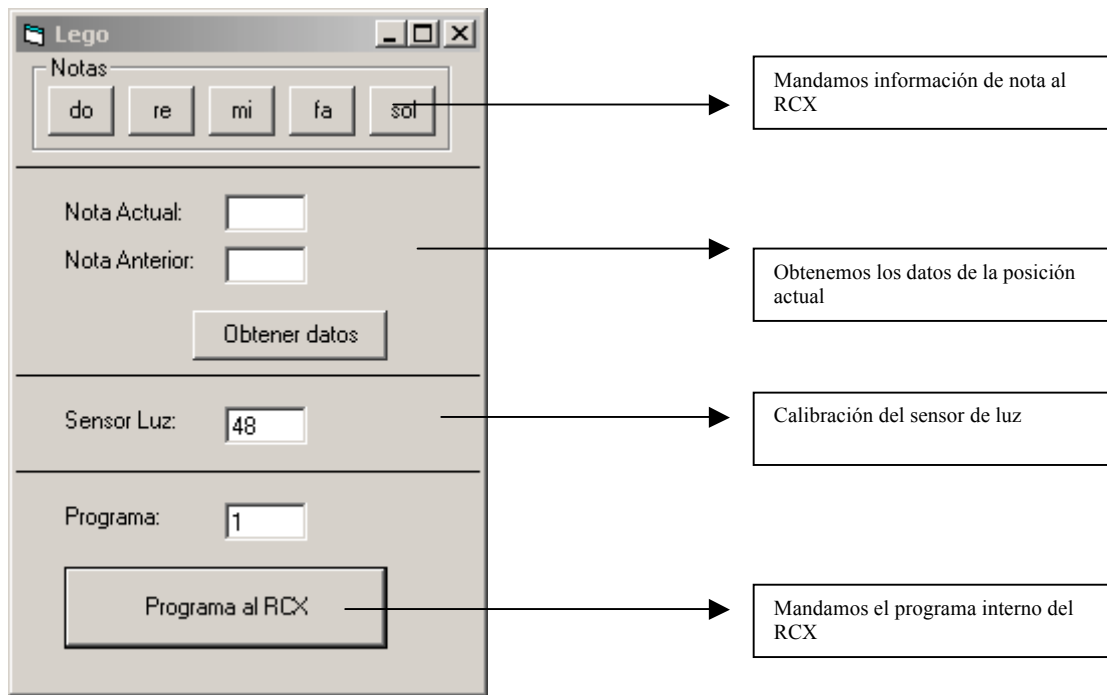
El formulario frmLego está dividido en cuatro partes.

La parte superior sirve para mandar información de nota al robot. Esto nos permite comprobar de forma manual que realmente el robot funciona correctamente, independientemente de la parte de reconocimiento de voz.

La segunda parte nos informa de la nota en la que se encuentra y de la nota anterior. Como veremos posteriormente el algoritmo para encontrar la nueva nota es bastante simple, basta con restar la posición objetivo de la posición actual y encontrar los pasos de nota correspondientes.

La tercera parte nos permite calibrar el sensor de luz. Éste es bastante sensible a los cambios de luminosidad de la habitación. Valores óptimos se encuentran entre 42 (habitación poco iluminada) y 50 (habitación bastante iluminada). Este valor indicará el umbral que determina el paso de nota.

El botón de la parte inferior transfiere el programa al RCX según el número elegido (1-5).



Las equivalencias del valor de las notas son las siguientes:

1	do
2	re
3	mi
4	fa
5	sol

Por tanto, simplemente con el siguiente cálculo obtenemos el número de pasos de nota. El signo indicará el sentido del motor

$$\text{PasosNotas} = \text{ValorActual} - \text{ValorAnterior}$$

En el RCX almacenamos 3 variables:

Var 1: ValorActual
 Var 2: ValorAnterior
 Var 3: PasosNotas

La variable 3 rige completamente el funcionamiento del robot. Esta puede tomar los siguientes valores:

100	robot parado
0	robot toca nota
1	robot adelanta 1 nota
2	robot adelanta 2 notas
.	.
.	.
-1	robot retrasa 1 nota
-2	robot retrasa 2 notas
.	.
.	.

El bucle principal del robot es el siguiente. (Fichero frmLego.frm):

```
'*****
'Bucle principal infinito
'*****
.Loop CTE, 0

'*****
'La variable 3 rige el comportamiento del robot
'Los valores posibles son
'100  robot parado
'0    robot toca nota
'1    robot adelanta 1 nota
'2    robot adelanta 2 notas
'.
'.
'-1    robot retrasa 1 nota
'-2    robot retrasa 1 nota
'.
'.
'*****

'*****
'Movimiento del robot para situarse en la nota correspondiente
'*****
.If VAR, 3, DIF, CTE, 100      'si var 3 no es 100
.If VAR, 3, MENQ, CTE, 0      'si variable 3 es menor que 0
    .SetRwd MOTOR_C           'motor atrás
.EndIf
.If VAR, 3, MAYQ, CTE, 0      'si variable 3 es mayor que 0
    .SetFwd MOTOR_C           'motor adelante
.EndIf
.On MOTOR_C                   'motor C on
.AbsVar 3, VAR, 3             'quitamos signo a la variable 3
.Loop VAR, 3                   'loop en función de var 3
    .Wait 2, 20               'esperamos 200ms, en ese tiempo no puede haber 2 pasos de nota
    .While SENVAL, SENSOR_3, MAYQ, CTE, txtSensorLuz.Text 'No hacemos nada mientras que el
    .EndWhile                  'de luz sea mayor que el umbral
    .EndLoop
.SetVar 3, CTE, 0             'Mandamos al robot a tocar nota poniendo var 3 igual a 0
.EndIf

'*****
'Movimiento del brazo del robot para tocar nota
'*****
.If VAR, 3, IG, CTE, 0        'si var 3 es 0
    .Off MOTOR_C               'paramos el motor de movimiento
    .SetFwd MOTOR_A            'encendemos el motor del brazo
    .On MOTOR_A
    .While SENVAL, SENSOR_1, IG, CTE, 0 'mientras no toquemos el sensor de colision
    .EndWhile
    .SetRwd MOTOR_A            'cambiamos el sentido de giro del motor
    .Wait CTE, 50              'dejamos medio segundo que suba el brazo
    .Off MOTOR_A               'paramos el motor del brazo
    .SetVar 3, CTE, 100        'variable 3 lista para recibir nueva nota
.EndIf
.EndLoop
```

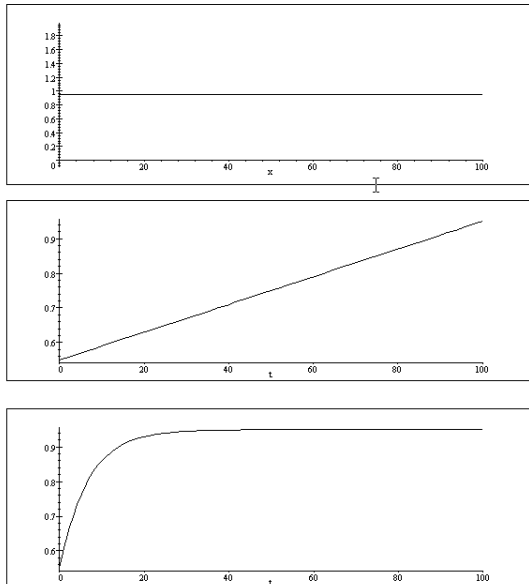
5.-Topología de la red.

Parámetros.

El sistema implementado es una red recurrente (RNN) formada por 20 células de entrada, 5 células ocultas y una célula de salida (20 x 5 x 1). La activación de las 5 células ocultas pasa a través de una unidad de retardo (z^{-1}) y se almacena en nodos de contexto. Debido a que la salida de los nodos de contexto está conectada a la entrada de cada célula en la capa oculta, éstos sirven como memoria de las activaciones de la capa oculta. La estructura se representa en la figura 13.

La salida de la red produce una salida en función del tiempo a medida que los nodos de entrada avanzan a través de los parámetros que representan la voz. La RNN se entrena usando el algoritmo “back-propagation”. Para cada palabra clave se representa una respuesta deseada a la salida de la red en el rango $[0,1]$. Los nodos de la capa oculta, $x_2^n(t=0)$, se inicializan con un valor de 0.5. La relación de aprendizaje fue un valor pequeño, $\eta = 0.15$ para evitar oscilaciones. Con estos datos y con tres muestras de voz por nota (3 x 5 (do-sol))=15, realizamos 250 iteraciones.

El sistema se entrenó con tres funciones objetivo. La que dio mejores resultados fue la exponencial.



$y(t) = y(m)$	Función escalón
$y(t) = \phi_y + \lfloor (y_m - \phi_y)t / T \rfloor$	Función rampa
$y(t) = \lfloor (y_m - \phi_y)e^{-15t/T} \rfloor$	Función exponencial

con los valores

$$y_m = 0.95 \quad \phi_i = 0.55$$

Si los datos de entrada son de rechazo, éstos se entrenan con $1-y(t)$.

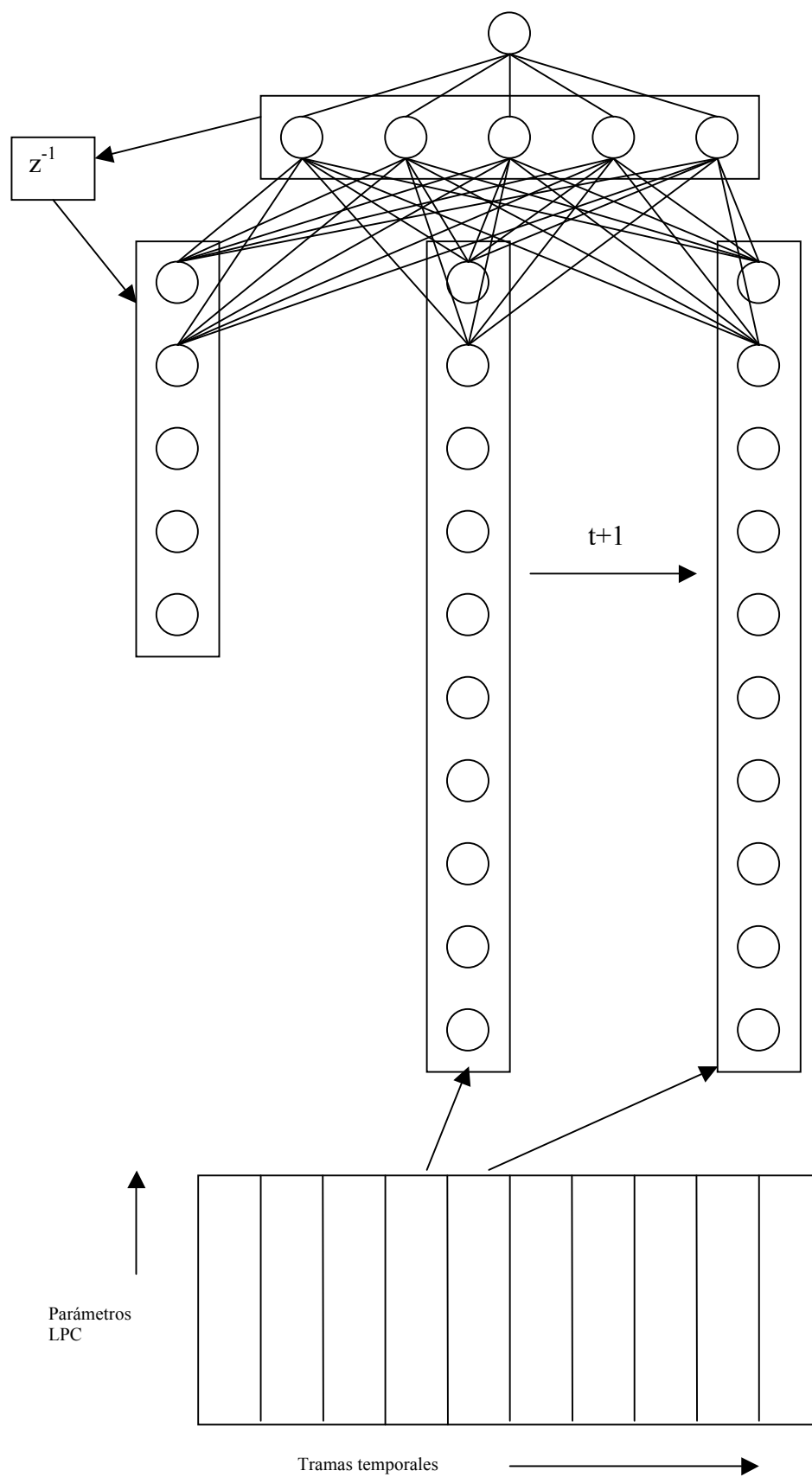


Fig.13

La entrada a la red está formada por dos vectores con datos consecutivos y una realimentación de la capa oculta. La ventaja de este tipo de entradas es que la red es capaz, directamente, de codificar diferencias en las características de las dos tramas consecutivas de vectores. Una red que procesara únicamente un vector cada vez, requeriría nodos de contexto adicionales para almacenar la información pasada y tendría una representación interna mas compleja.

Morgan y Scofield [Morgan 91] investigaron el número óptimo de nodos en la capa oculta para redes con la estructura aquí presentada. Sus investigaciones revelan que para palabras como “westchester”, los resultados no mejoran con una capa oculta de mas de cinco nodos. Esto nos llevó a implementar este dato como óptimo en la capa oculta de nuestra red

Implementación software

La red de reconocimiento fue implementada en lenguaje C empleando el compilador Visual C 6.0 de Microsoft para generar dos ficheros .dll que son entrenam.dll y verific.dll (redes de entrenamiento y verificación). Los motivos de emplear C, a pesar de su complicación, son evidentes:

- Rapidez. En este lenguaje podemos manejar un gran número de datos con velocidades realmente altas. El sistema de entrenamiento es principalmente iterativo (bucles), por lo cual, implementar la red de entrenamiento en un programa matemático (matlab, mathematica...) o en un lenguaje de programación más lento (Visual Basic o incluso JAVA) incrementaría el tiempo de trabajo considerablemente.
- Portabilidad: el hecho de escribir la red en ficheros dll con funciones que pueden ser llamadas desde el exterior, permite que desde otro lenguaje de programación mas sencillo (ej. Vbasic) se establezca la interfaz de usuario, encapsulando la complejidad en C. El inconveniente de usar un programa matemático o un simulador es la necesidad de trabajar siempre dentro de ese programa y no con un .exe independiente. Además, las funciones empleadas (fft.cpp, lpc.cpp, hamming.cpp...) pueden ser reutilizadas por cualquier persona con conocimientos en C++. El C empleado es estándar ANSI-C que tiene la ventaja de poder ser exportado a otros sistemas operativos distintos de Microsoft windows

Todos los datos del tipo $N_{b,c,d}^a$ los hemos implementado en estructuras matriciales de la forma N[a][b][c][d]. Esto tiene el inconveniente de una pérdida considerable de memoria debido a la necesidad de reservar un espacio a*b*c*d datos que no siempre son necesarios. La ventaja, evidentemente, es la facilidad de programación. Hay que destacar también que el tipo de datos empleado es ASCII. Para todas las operaciones de lectura y escritura en fichero se emplean datos ASCII separados por retorno de carro. De esta manera facilitamos la extracción y lectura de datos por cualquier programa matemático u hoja de cálculo. (Los datos pueden visualizarse directamente en una hoja de calculo excel de Microsoft)

6.- Manual de usuario

Una vez ejecutado el programa principal, tras la instalación, veremos una pantalla como la siguiente.Figura 14.

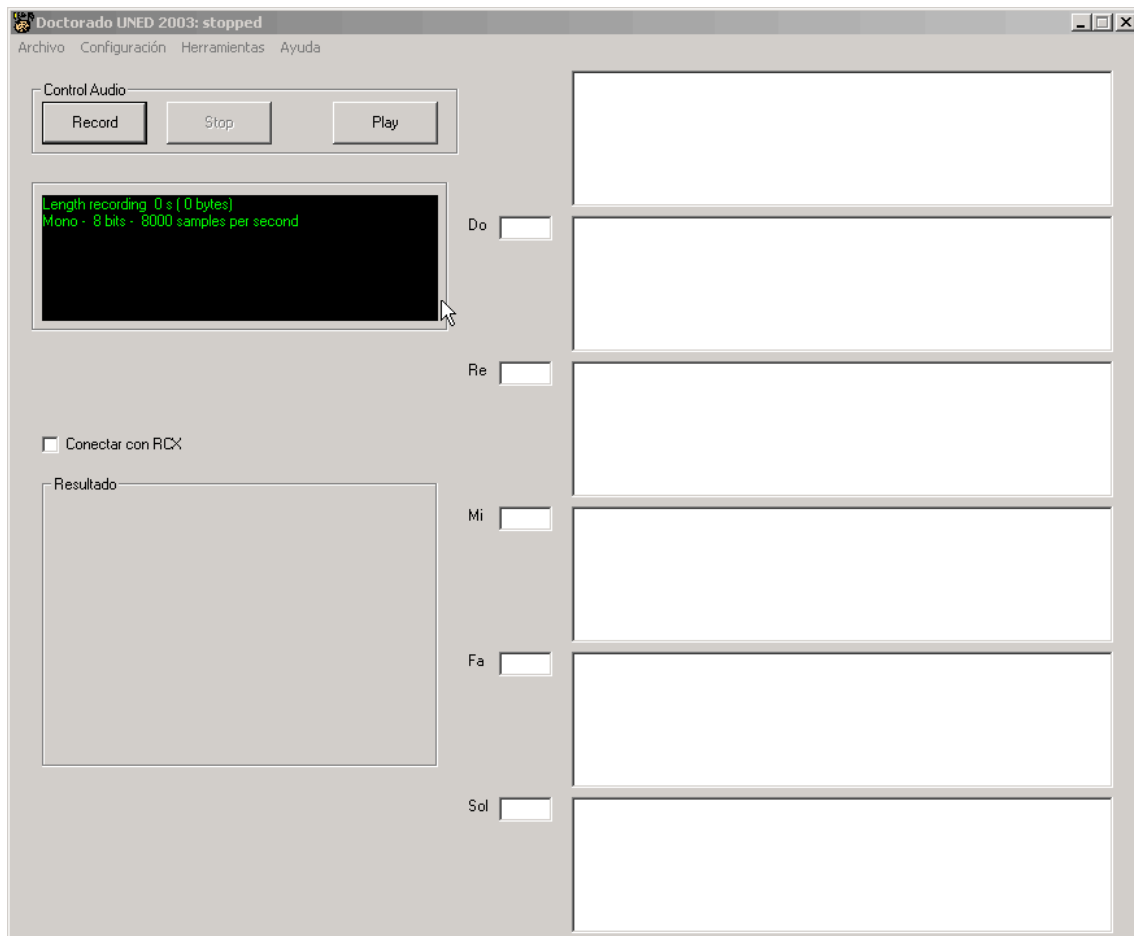


Fig.14

Las partes mas importantes del programa son:

- Si en el menú pulsamos Configuración>Audio... nos aparece una ventana que nos permitirá cambiar la frecuencia de muestreo, el número de canales y la resolución.

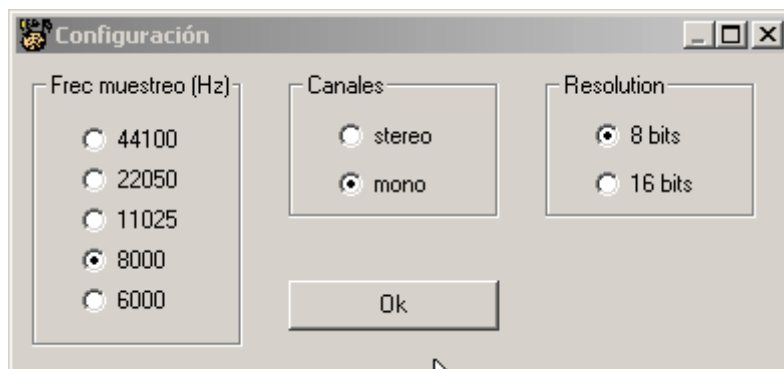


Fig. 15

Por defecto está puesta la frecuencia de muestreo a 8000, mono y 8 bits, pero sería posible configurarlo para que el sistema trabajase con una resolución mayor. En todo el proyecto se ha trabajado con estos

datos para tomar como referencia una calidad telefónica, suficiente para la inteligibilidad, que es lo que en estos momentos nos interesa.

- Pulsando Configuración>Lego... aparece la siguiente ventana. Esta es la parte de comunicación con el robot y nos remitimos a lo comentado en el punto 4.

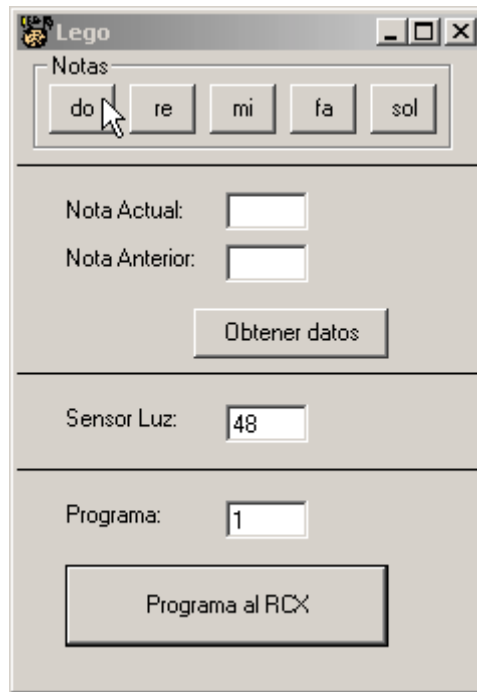


Fig. 16

- Si pulsamos en el menú Herramientas>Entrenamiento... nos aparecerá una ventana en la que podemos configurar el entrenamiento

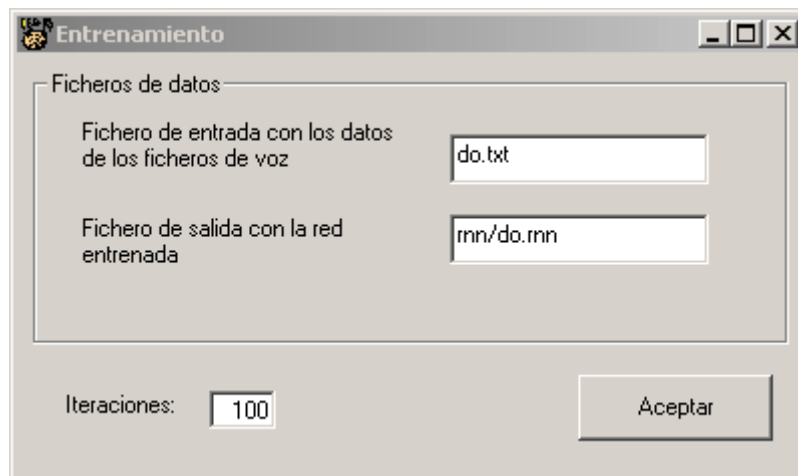
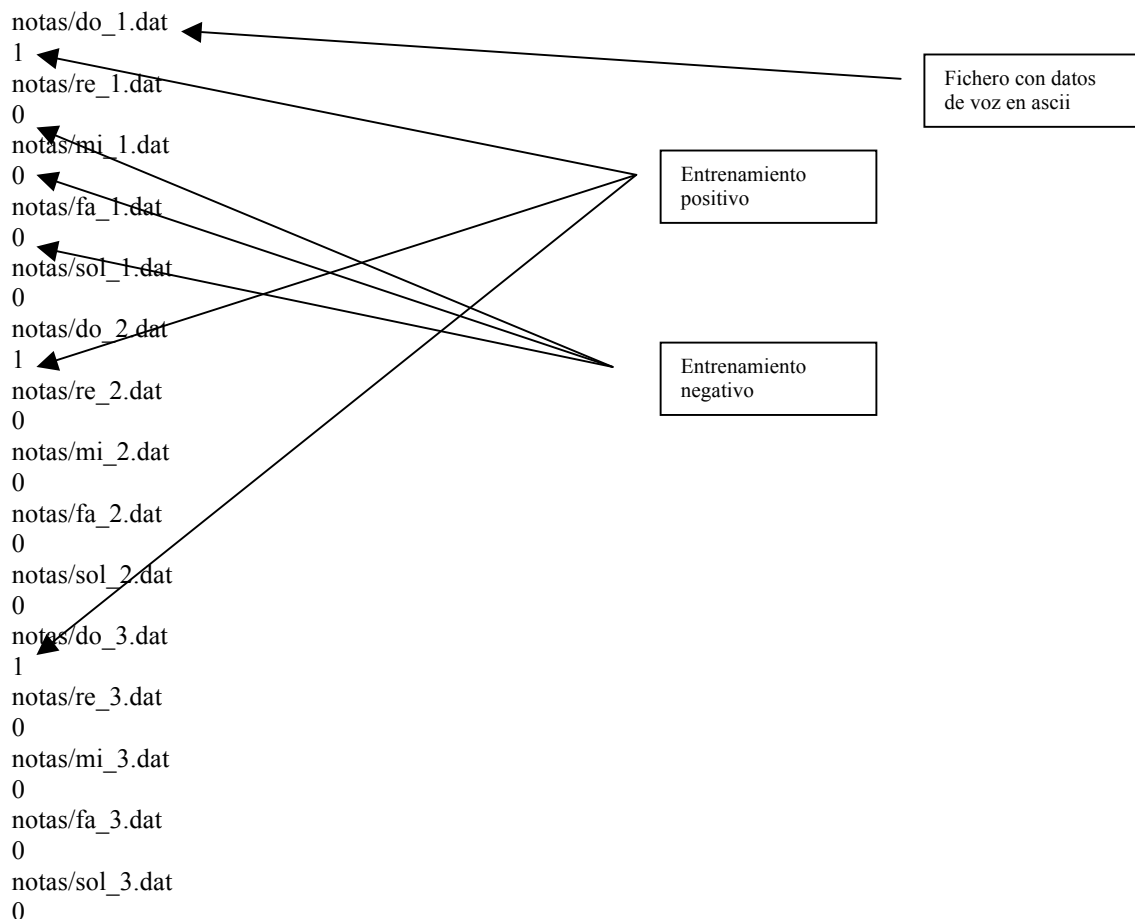


Fig. 17

La primera caja de texto nos pregunta el fichero con los datos de entrada. Este fichero debe tener una estructura similar a la siguiente:



Separados por retorno de carro en el fichero de texto se encuentra el path de los ficheros de audio para entrenar. Debajo del fichero, con un 1 o un 0 indicamos si el entrenamiento a realizar para es fichero será positivo o negativo, es decir, si queremos que reconozca o no esa nota musical. Este proceso se debe realizar 5 veces, una por cada nota.

En la segunda caja de texto introducimos el path de la red de salida. Los ficheros en los que se encuentran los datos de la red son los *.RNN. La estructura de estos ficheros está formada por un valor de cabecera (V) y los datos en ascii correspondientes al valor de los pesos de las conexiones.

En la última caja de texto indicamos el número de iteraciones para el entrenamiento. El tiempo que tarda el sistema en realizar el entrenamiento es proporcional al número de iteraciones. Sobre un pentium IV a 2.5GHz, un entrenamiento de 250 iteraciones de 15 ficheros de voz tarda aproximadamente 20 segundos.

Las muestras de voz deben estar correctamente cortadas para el entrenamiento. Para ello hemos incluido una utilidad de edición de ficheros de audio al que podemos acceder bien mediante el botón Editar, bien mediante el menú Archivo>Editar. De esta manera podemos segmentar cómodamente las palabras para ser entrenadas por el sistema. Al pulsar “Guardar selección” almacenamos en dos ficheros (.wav, .dat) la selección de la onda en el path indicado en la caja de texto inferior, es decir si en la figura 18 pulsamos “Guardar selección”, la selección del fichero de onda se guardará en “notas/do.dat” como fichero asii y en “notas/do.wav” como fichero de sonido.

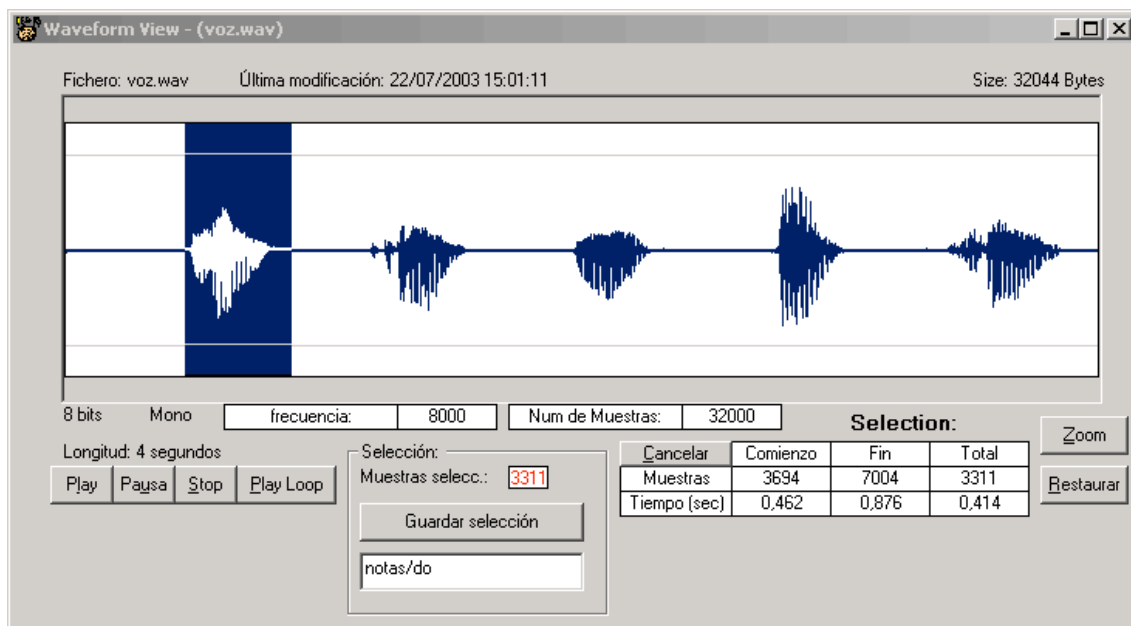


Fig 18

Una vez entrenada la red, podemos probar el sistema, para ello, si pulsamos Record, el programa grabará los datos de la entrada predeterminada de la tarjeta de sonido, almacenándolos en un doble formato:

- voz.wav: fichero de audio
- voz.dat: fichero de audio con los datos en ascii.

Una vez grabado el fichero, el sistema automáticamente lo pasa a través de las 5 redes entrenadas y se visualiza el resultado. En la figura 18 se muestra el resultado obtenido al grabar "do" en el micrófono. En la primera de las cajas de imagen vemos la señal en tiempo y en las otras 5 la salida de cada una de las redes. Como podemos observar, el sistema ha reconocido claramente que se trata de la palabra "do". En este caso, la "o" intenta excitar la salida de la red "sol", pero pierde claramente en la competencia frente a la red "do"

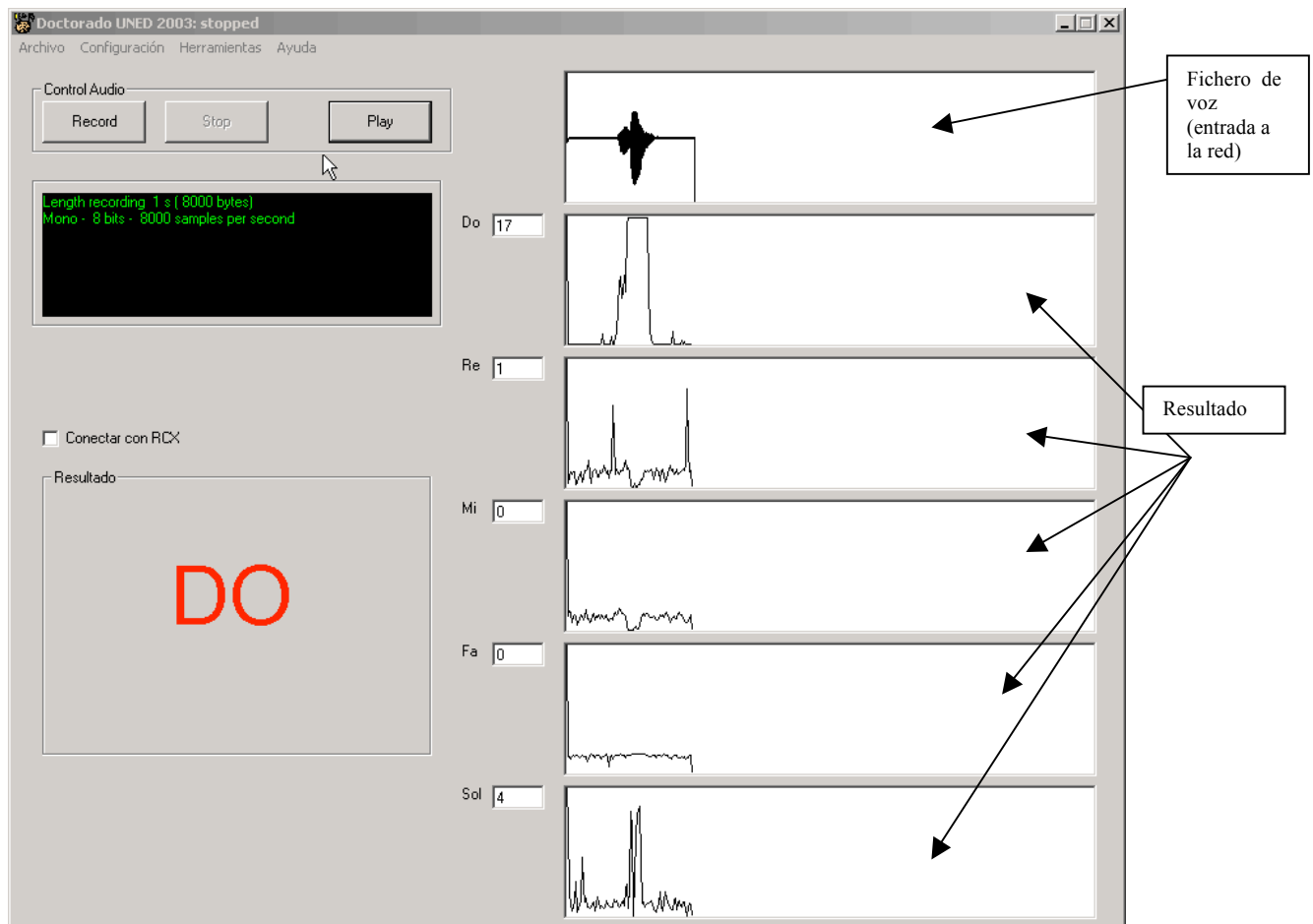


Fig. 18

Para analizar los datos mediante un programa matemático u hoja de cálculo, los datos de salida quedan almacenados en 5 ficheros en la carpeta resultados: salida1.dat, salida 2 .dat, ... salida5.dat

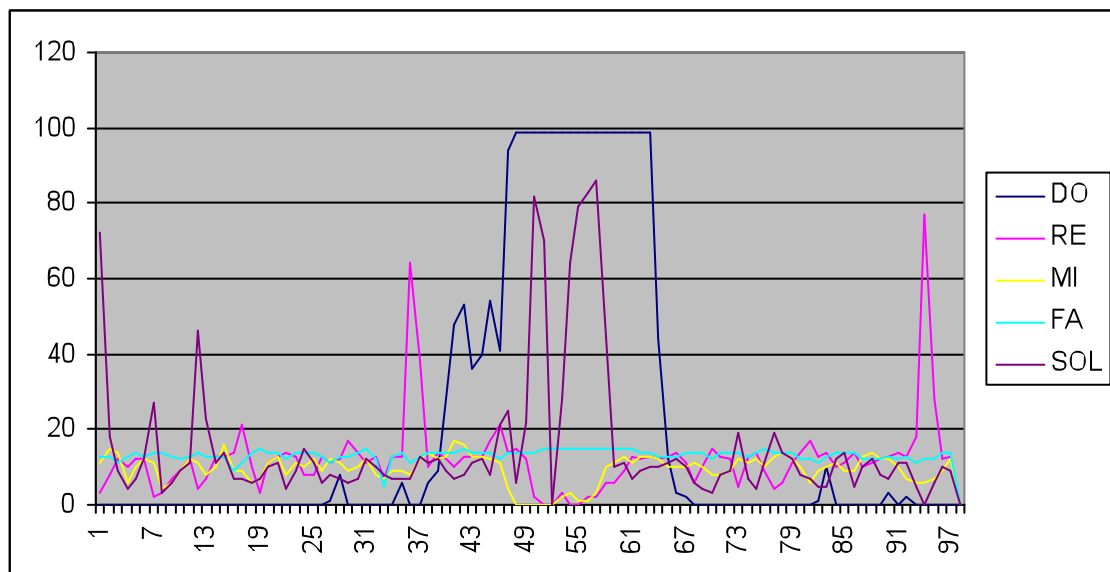


Fig. 19

si pulsamos la opción de “conectar con RCX”, directamente la información de nota será transmitida al robot LEGO para que ejecute la nota correspondiente.

Haciendo pruebas mas exhaustivas, hemos obtenido los siguientes resultados.

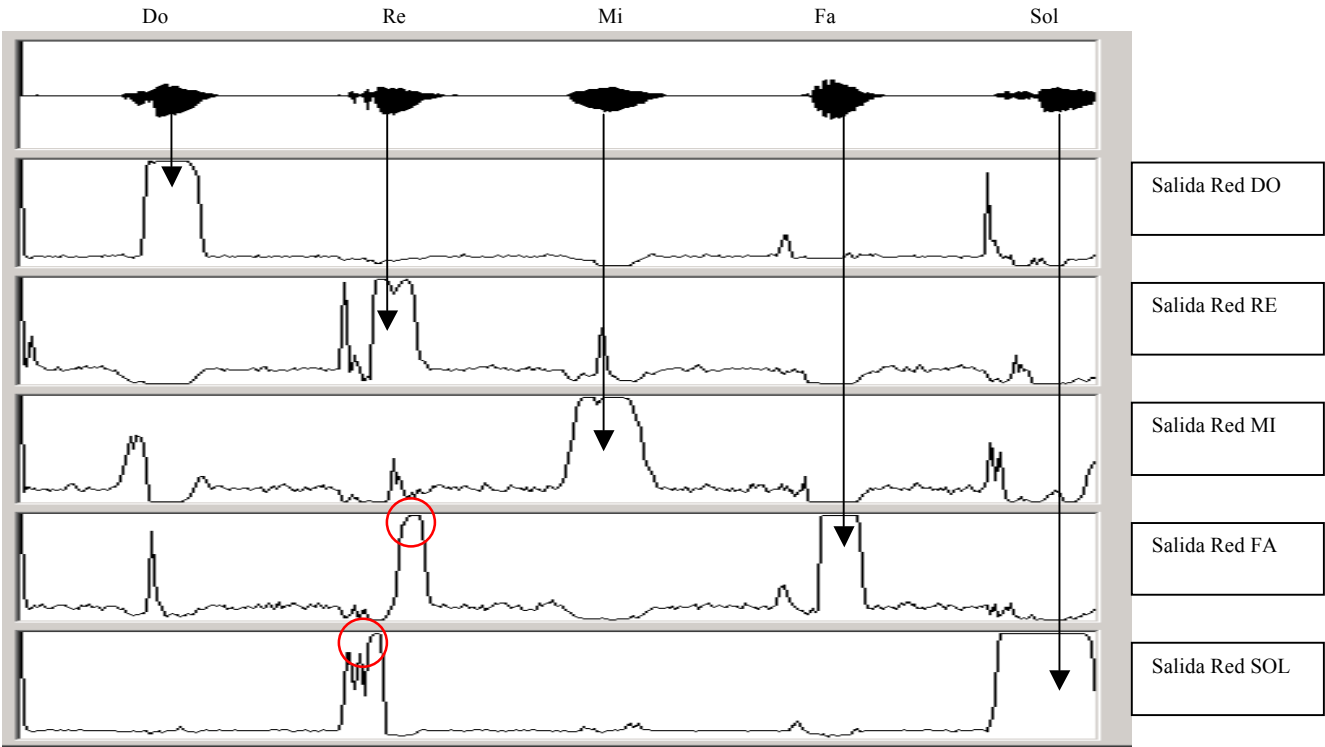


Fig. 20

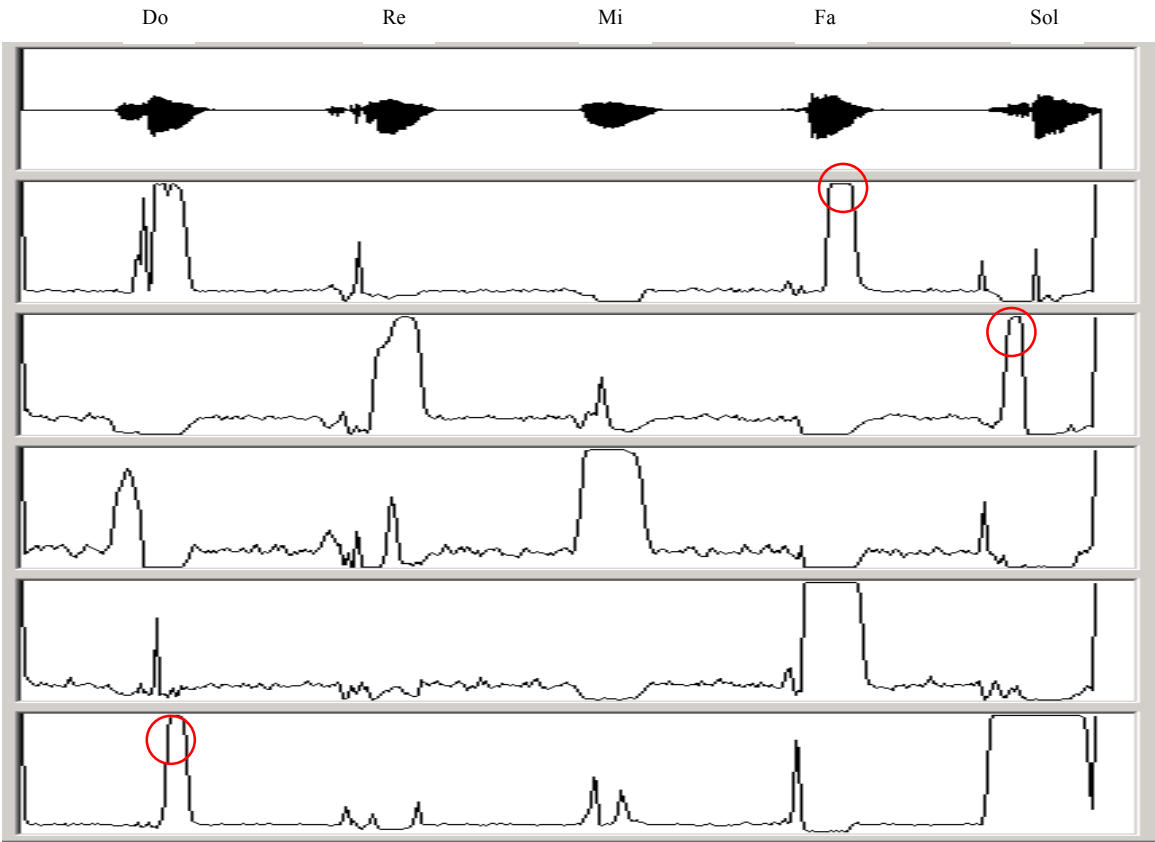
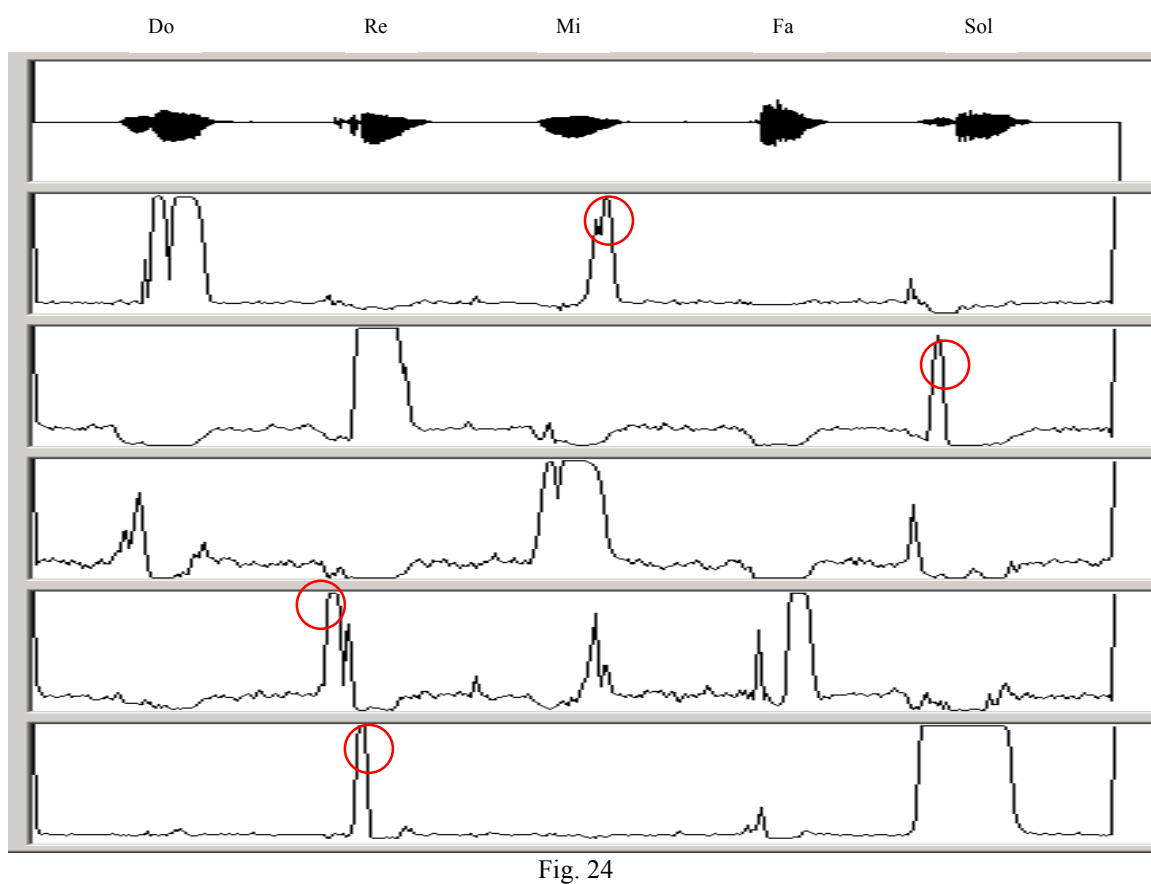
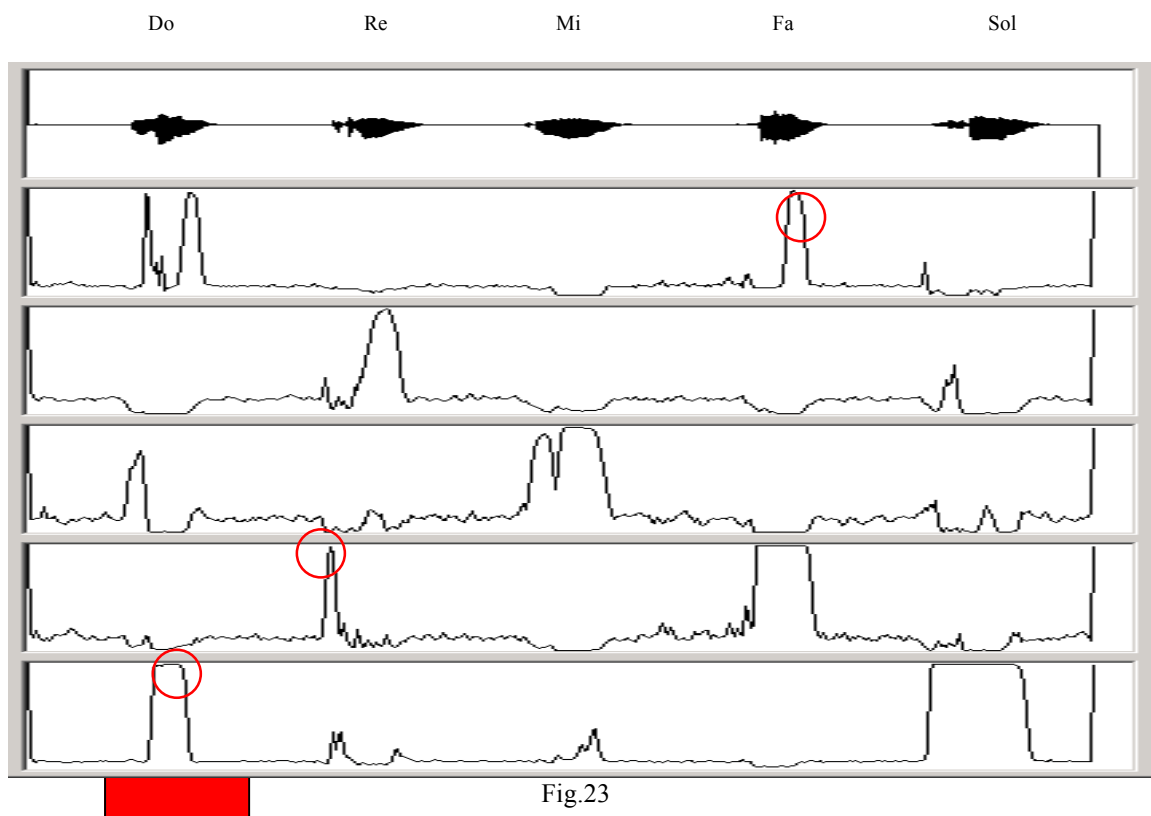


Fig. 21



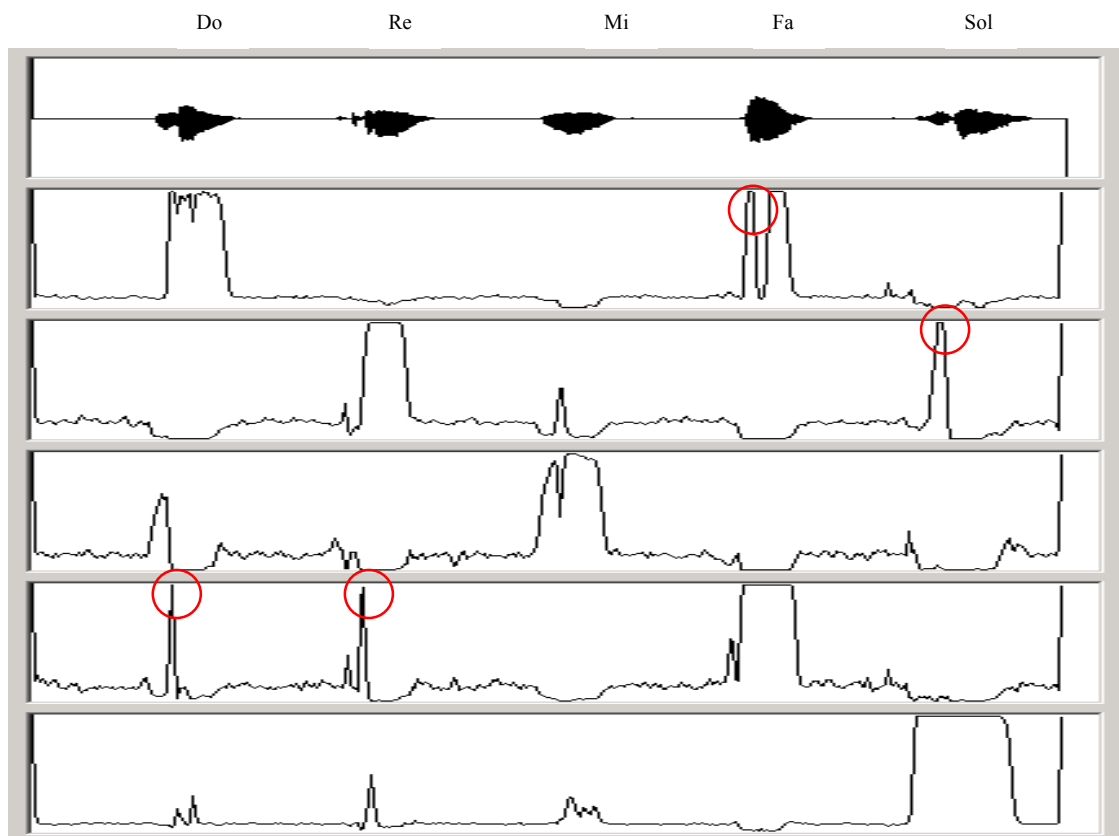


Fig. 25

Hemos señalado con un círculo rojo los posibles errores de activación de cada red, si actuaran de forma independiente. Como podemos observar, en el momento en que se produce la competencia entre las redes de salida, disparada por un umbral de energía de la señal de voz, el resultado de acierto se encuentra en torno al 95%. De las 25 muestras de voz tomadas, sólo hay un caso de error en la salida (respuesta “sol” con la sílaba “do”). Está marcado por un rectángulo rojo en la figura 23.

7.- Conclusiones.

En este trabajo hemos realizado un sistema limitado de reconocimiento de voz por redes neuronales realimentadas, cuya salida es aplicada a un robot para que realice una tarea. Los resultados obtenidos han sido bastante satisfactorios. Separando ambas partes, podemos afirmar que el reconocimiento de voz está en torno al 95 % de acierto y el robot lleva a cabo su tarea prácticamente con un 100% de efectividad. Evidentemente, las tareas impuestas en ambas partes eran limitadas. El sistema de reconocimiento tiene un vocabulario limitado de 5 palabras, dependiente del hablante y palabras separadas por silencio, y el robot necesita ser situado en su posición inicial, desplazándose en una dirección y dos sentidos. Aún así pensamos que el trabajo realizado puede ser un punto de partida para un trabajo mas extenso y ambicioso de reconocimiento de voz con implicaciones prácticas.

A la vista de los resultados, podemos afirmar, que este tipo de estructuras realimentadas pueden ser empleadas con gran acierto formando parte de sistemas más complejos encargados de reconocimiento de palabras y frases conectadas. Figura 20.

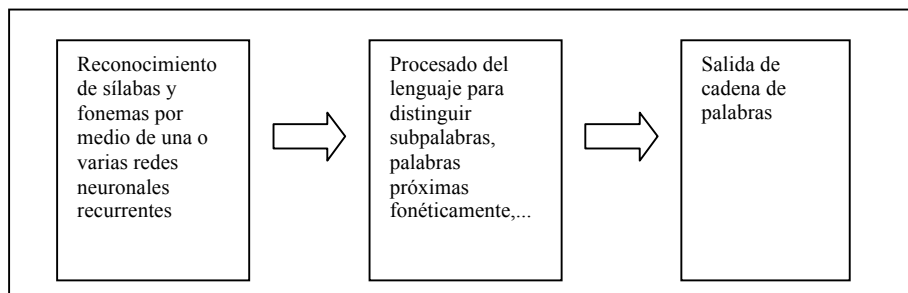


Fig.20

En realidad, esta forma de trabajar podría parecerse al funcionamiento del cerebro humano. Obtenemos información de una serie de sonidos y fonemas y, con el módulo de procesado del lenguaje, intentamos que tome sentido para formar palabras y frases.

Otras técnicas a tener en cuenta en una futura ampliación del proyecto serían Dinamic Time warping (DTW) y Modelos Ocultos de Markov (HMM), no comentados en este trabajo por salirse del ámbito de las redes neuronales, así como el conocido trabajo de kohonen del mapa fonotópico [Kohonen 88].

En el caso del robot, una primera ampliación sería la detección de la posición inicial por medio de una web-cam, estableciendo un matching entre un grid de una imagen en memoria y un grid de una imagen capturada con la cámara. Esto posiblemente daría buen resultado teniendo en cuenta la disposición en teclas negras y blancas de un teclado.

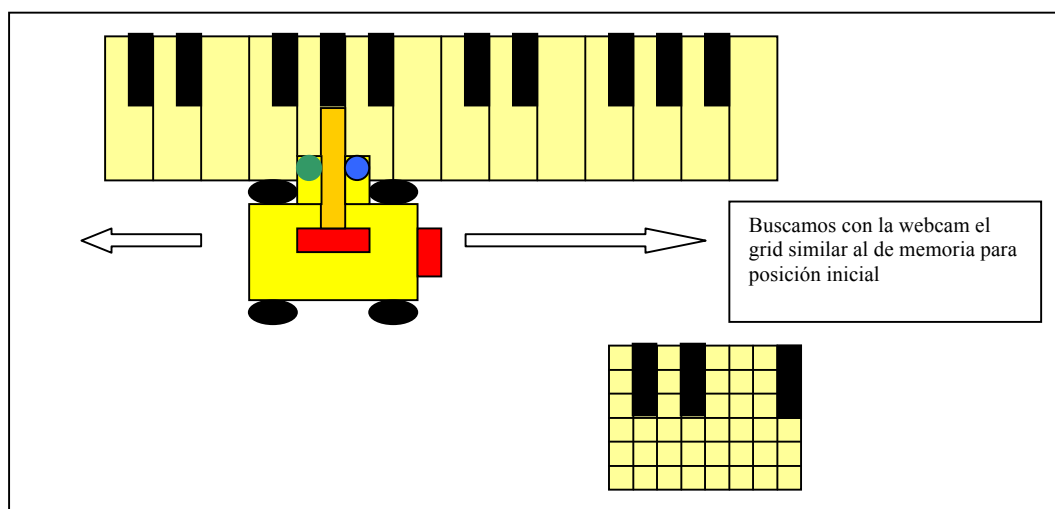


Fig. 21

Una segunda ampliación, que el robot llegase hasta el teclado y se colocase paralelo al mismo para buscar la posición inicial. Para esto sería necesario un segundo motor y al menos dos sensores de infrarrojos. Si suponemos que el robot se encuentra en una mesa sin mas obstáculos que el teclado, por medio de dos sensores de infrarrojos podemos colocar el robot de forma paralela al mismo, con una distancia determinada, listo para buscar la posición inicial con la webcam.

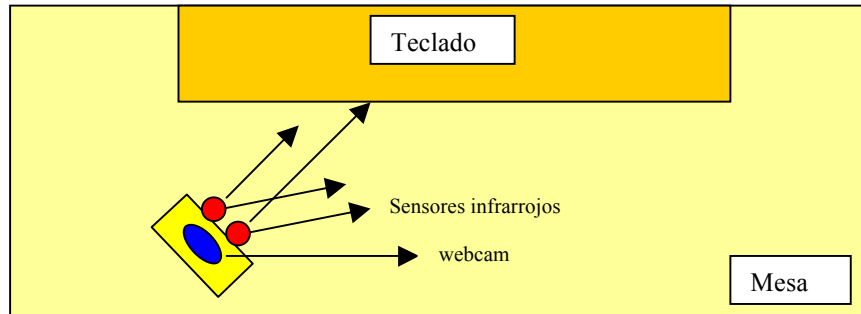


Fig.22

8.- Bibliografía.

- [Alexander 90] I. Alexander H. Morton, "An introduction to neural Computing", Chapman Hall, 1990
- [Arkin 98] Ronald C. Arkin. "Behavior Based Robotics", MIT Press, 1998.
- [Bagnan 02] Brian Bagnall, "Lego mindstorms programming", prentice hall, 2002.
- [Bekey 93] G.A. Bekey, K.Y. Goldberg, "Neural Networks in Robotics", Kluwer Academic Publishers, 1993
- [Bernal 00] J. Bernal, J. Bobadilla, P. Gómez, "Reconocimiento de voz y fonética acústica", Ra-Ma, 2000
- [Brío 01] B. Martín del Brío, A. Sanz Molina, "Redes neuronales y Sistemas Borrosos", Ra-Ma, 2001
- [Ceballos 91] Fco. Javier Ceballos, "Enciclopedia del lenguaje C", Ra-Ma, 1991
- [Ceballos 96] Fco. Javier Ceballos, "Enciclopedia de Visual Basic" Ra-Ma, 1996
- [Ceballos 02] Fco. Javier Ceballos, "Microsoft Visual C++", Ra-Ma, 2002
- [Connell 93] J.H. Connell, S. Mahadevan, « Robot learning », Kluwer Academic Publishers, 1993
- [Dayhoff 90] Judith E. Dayhoff, "Neural Networks Architectures", Van Nostrand Reinhold, New York, 1990
- [Furui 89] Sadaoki Furui, "Digital Speech Processing, Synthesis, and Recognition", Marcel Dekker, INC, 1989
- [Hilera 95] J.R. Hilera, V.J. Martínez, "Redes Neuronales Artificiales", Ra-Ma, 1995
- [Hush 92] Don R. Hush y Bill Horne, "An overview of neural networks", Informática y Automática. Vol 25. 1992
- [Jackson 89] Leland B. Jackson, "Digital Filters and Signal Processing", Kluwer Academic Publishers, 1998
- [Latombe 91] J.C. Latombe, "Robot Motion Planning", Kluwer Academic Publishers, 1991
- [Lippmann 87] Richard P. Lippmann, "An Introduction to Computing with Neural Nets", IEEE ASSP Magazine, April 1987
- [Microsoft 01] "Microsoft Visual Basic 6.0", Microsoft Press, 2001
- [Morgan 91] D.P. Morgan, C.L. Scofield, "Neural Networks and Speech Processing", Kluwer Academic Publishers, 1991
- [Recuero 83] Manuel Recuero López, "Acústica", EUITT, 1983
- [Shannon 48] C.E. Shannon, "A mathematical theory of communication", Bell System technical Journal, vol27, 1948
- [Shaughnessy 90] Douglas O'Shaughnessy, "Speech Communication Human and Machine", Addison-Wesley Publishing Company, 1990
- [Watrous 87] R.L. Watrous and L. Shastri, "Learning phonetic features using connectionist networks: An experiment in speech recognition", IEEE First Int. Conf on Neural networks, San Diego, junio 89, vol IV

9.- Código Fuente

entrenam.dll

Entrenam2003.cpp

```
/*
*****
/* -----Metodo de entrenamiento de una red recurrente de 3 capas ----- */
/* -----(version ANSI)----- */
/*----- */
/* Fichero "main" : ENT_ANSI.CPP */
/* Funciones a llamar : */
/* HAMMING.CPP */
/* LPC.CPP */
/* DAT_ASC.CPP */
*****

#include "stdafx.h"
#include<stdlib.h>
#include<stdio.h>
#include<math.h>
#include<conio.h>
// #include <alloc.h>

/* ----- declaraci3n de funciones ----- */

float (*lpc(float X[],int polos));
float (*imp_dat_ascii(char nombre[30]));
float (*hamming(int ));
float (*lpccep(float X[],int polos));

/* ----- */
/* ----- comienzo de la funcion entrenamiento ----- */
/* ----- */

bool _stdcall EntrenamRNN(char* argv1,int argv2, char* argv3);
bool _stdcall EntrenamRNN(char* argv1 ,int argv2,char* argv3)
{
/*----- declaramos variables----- */
static int pant,n_fic;
static float error=0,sum_error=0;
static float x_nod[4][6][11]; /* matriz de nodos */
static float w[4][6][4][11]; /* matriz de pesos */
static float w_inc[4][6][4][11]; /* matriz de incremento de pesos */
static float increm=0;
static float w_sal[156]; /* array de salida de pesos */
static int dimen_array=0; /* dimensiones del array de datos*/
static float *data_bin,*ham; /* puntero del array de datos */

static int l1,l2,l3,l4,z; /* variables para bucles */
static char buffer1[20];
static char *nom_cont;

static float x_nod2[6]={0,0,0,0,0,0};
static float delta[4][6][11];
FILE *pf,*pfl;
int conv=0,i=0,r=0;
static float S=0,S11[6]={0,0,0,0,0,0};
int ta,ca;
static float X1[240],X2[240];
static float *punt1,*punt2;
float sum=0;
static float resp_des=0;
static float xs[200];
static float resd[200];
struct ficha
{
char nombref[30];
int entf;
float mayorf;
};
struct ficha fich_dat[200];
int num_fich;
```

```

        int cont;
        float valor;

/* ----- Opci3n de tipo de entrenamiento ----- */
/* ----- Opci3n de tipo de entrenamiento ----- */
/* ----- Opci3n de tipo de entrenamiento ----- */

char* fdat;
fdat=argv1;

if ((pfl=fopen(fdat,"r"))==NULL)
{
    printf("El fichero %s no se puede abrir",fdat);
    exit(0);
}
char cadena[30];
z=1;
while(!feof(pfl))
{
    fscanf(pfl,"%s",fich_dat[z].nombref);
    fscanf(pfl,"%s",cadena);
    if (fich_dat[z].nombref==" ")
    {
        break;
    }
    fich_dat[z].entf=atoi(cadena);

    if ((pf=fopen(fich_dat[z].nombref,"rt"))==NULL)
    {
        printf("\nEl fichero %s no se puede abrir.",fich_dat[z].nombref);
        fclose (pf);
        exit(0);
    }
    fich_dat[z].mayorf=0;
    while(!feof(pf)) /* Calculamos el rango de altura de datos */
    {
        fgets(buffer1,20,pf);
        valor=atof(buffer1);
        cont=cont+1;
        if (fabs((double)valor)>fich_dat[z].mayorf)
        {
            fich_dat[z].mayorf=fabs((double)valor);
        }
    }

    fclose(pf);
    num_fich=z;
    z=z+1;
}

//fclose(pfl);

/* ----- Opcion de continuar o comenzar entrenamiento ----- */
/* ----- Opcion de continuar o comenzar entrenamiento ----- */
/* ----- Opcion de continuar o comenzar entrenamiento ----- */

principio:

for(l1=0;l1<=3;l1++)
{
    for (l2=0;l2<=5;l2++)
    {
        for (l3=0;l3<=3;l3++)
        {
            for (l4=0;l4<=10;l4++)
            {
                w[l1][l2][l3][l4]=(rand())/32767.0-0.5)/2.0;
            }
        }
    }
}

```

```

    }
    }
}

/*-----*/
/* ----- introducimos un valor de 0.5 en la capa 2 ----- */
/* -----*/

    for (l1=1;l1<=5;l1++)
    {
        x_nod[2][l1][0]=0.5;
    }

/* -----*/
/* ----- N mero de iteraciones ----- */
/* -----*/

int iter;
printf("\n N mero de repeticiones para la convergencia: ");

iter=argv2;

/* Nombre del fichero de salida */

FILE *ptabla;
char *fichero;
printf("\n Introduzca el nombre del fichero.rnn (salida): ");

fichero=argv3;

    for (conv=1;conv<=iter;conv++)
    {

/* -----*/
/* ----- llamamos a la funci n datos ----- */
/* -----*/

for (n_fic=1;n_fic<=num_fich;n_fic++)
{

    data_bin=imp_dat_ascii(fich_dat[n_fic].nombref);
    dimen_array=data_bin[9999]; // n mero de datos.
    ham=hamming(240);

/*-----*/
/* -----*/
/* ----- BUCLE PRINCIPAL----- */
/* -----*/
/*-----*/

    for (ta=0;ta<=dimen_array-320;ta+=80) /* llamamos a todos los datos del array */
    {
        extern void clrscr();

printf("Iteraci n=%d\nfichero(%d)=%s\nmuestra=%d",conv,n_fic,fich_dat[n_fic].nombref,ta)
;

        for (ca=0;ca<=239;ca++) /*recogemos 2 matrices de
datos para*/
        {
            /*llevarlos a lpc y obtener 2 tramas */
            X1[ca]=((data_bin[ca+ta]/fich_dat[n_fic].mayorf)*ham[ca]*1.0); /* de
par metros */
            X2[ca]=((data_bin[ca+ta+80]/fich_dat[n_fic].mayorf)*ham[ca]*1.0);
        }

/* -----*/
/* ----- llamamos a la funci n lpc ----- */
/* -----*/

        punt1=lpc(X1,10);

```

```

punt2=lpc(X2,10);

/* colocamos los correspondientes valores lpc en sus correspondientes
   tramas */

for (l3=1;l3<=10;l3++)
{
    x_nod[1][2][l3]=punt1[l3];
    x_nod[1][3][l3]=punt2[l3];
}

/* ----- Comenzamos el c lculo ----- */
/* ----- Comenzamos el c lculo ----- */
/* ----- Comenzamos el c lculo ----- */

for (l1=6;l1<=10;l1++)
{
    /* nodos vacios */
    x_nod[1][1][l1]=0.0;
}

/* ----- C lculo de funcion sigmoidal capa de salida ----- */

S=0.0;
for(z=1;z<=5;z++)
{
    sum=x_nod[2][z][0]*w[2][1][z][0];
    S=S+sum;
}
x_nod[3][0][0]=1.0/(1.0+exp((double)(-S))); /* actual salida de red */

/* ----- comenzamos a adaptar los pesos de las capas (adapt weights) ----- */
/* ----- comenzamos a adaptar los pesos de las capas (adapt weights) ----- */
/* ----- comenzamos a adaptar los pesos de las capas (adapt weights) ----- */

/*****
/* respuesta deseada */
*****/
//función exponencial para respuesta deseada

if (fich_dat[n_fic].entf==1)
{
    resp_des=0.95-((0.95-0.55)*exp((double)(-15.0*ta/dimen_array)));
}
else if (fich_dat[n_fic].entf==0)
{
    resp_des=1-(0.95-((0.95-0.55)*exp((double)(-
15.0*ta/dimen_array))));
}

/* ----- adaptamos capa de salida ----- */
/* ----- adaptamos capa de salida ----- */
/* ----- adaptamos capa de salida ----- */

delta[3][0][0]=(x_nod[3][0][0]*(1.0-x_nod[3][0][0]))*(resp_des-x_nod[3][0][0]);

/* ----- adaptamos 2ª capa intermedia ----- */
/* ----- adaptamos 2ª capa intermedia ----- */
/* ----- adaptamos 2ª capa intermedia ----- */

for (i=1;i<=5;i++)
{
    delta[2][i][0]=(delta[3][0][0]*w[2][1][i][0])*(x_nod[2][i][0]*(1-
x_nod[2][i][0]));
}

/* ----- Adaptamos 1ª capa ----- */
/* ----- Adaptamos 1ª capa ----- */
/* ----- Adaptamos 1ª capa ----- */

S=0;
for (l1=1;l1<=3;l1++)
{
    for (l2=1;l2<=10;l2++)
    {
        for (r=1;r<=5;r++)
        {

```

```

        sum=delta[2][r][0]*w[1][r][l1][l2];
        S=S+sum;
    }
    delta[1][l1][l2]=S*x_nod[1][l1][l2]*(1.0-x_nod[1][l1][l2]);
    S=0;
}
}

/*****
/* ----- Calculamos los nuevos promediados w (weights) ----- */
/*****
/*****
/* w capa 1 */
/*****

for (l1=1;l1<=5;l1++)
{
    for (l2=1;l2<=3;l2++)
    {
        for (l3=1;l3<=10;l3++)
        {
            increm=(0.15*delta[2][l1][0]*x_nod[1][l2][l3])+0.05*w_inc[1][l1][l2][l3];
            w[1][l1][l2][l3]=w[1][l1][l2][l3]+increm;

            w_inc[1][l1][l2][l3]=increm;
        }
    }
}

/* -----
/* ----- Guardamos los antiguos x_nod de la capa 2 ----- */
/* -----

for (z=1;z<=5;z++)
{
    x_nod2[z-1]=x_nod[2][z][0];
}

/* -----
/* ----- Calculamos los nuevos x_nod de capa 2 ----- */
/* -----

for (z=0;z<=5;z++)
{
    S11[z]=0.0;
}

for (z=1;z<=5;z++)
{
    for (l1=1;l1<=3;l1++)
    {
        for (l2=1;l2<=10;l2++)
        {
            sum=x_nod[1][l1][l2]*w[1][z][l1][l2];
            S11[z]=S11[z]+sum;
        }
    }
    x_nod[2][z][0]=(1.0/(1.0+exp(((double)(-S11[z]))))); /* tomando B=1 */
}

/* -----
/* ----- Calculamos los nuevos w de capa 2 ----- */
/* -----

for (l1=1;l1<=5;l1++)
{
    increm=(0.15*delta[3][0][0]*x_nod[2][l1][0])+0.05*w_inc[2][1][l1][0];
    w[2][1][l1][0]=w[2][1][l1][0]+increm;
    w_inc[2][1][l1][0]=increm;
}

/* -----
/* -----Calculamos el valor de salida ----- */
/* -----

```

```

S=0.0;
for(z=1;z<=5;z++)
{
    sum=x_nod[2][z][0]*w[2][1][z][0];
    S=S+sum;
}
x_nod[3][0][0]=1.0/(1.0+exp(((double)(-S)))); /* actual salida de red */

/* ----- */
/* ----- Retardo Z-1 de la capa 2 a entrada 1 ----- */
/* ----- */

    for (z=1;z<=5;z++)
    {
        x_nod[1][1][z]=x_nod2[z-1];
    }

/* Representamos la capa intermedia */

if(kbhit()!=0)
{
    pant=getch();
}

/* Opci3n de abandonar */

if (pant==0x71)
{
    char ex;
    //clrscr();
    printf("\nDesea abandonar el proceso (S/N)? ");
    if (ex==0x73)
    {
        fclose(pf);
        exit(0);
    }
    fflush(stdin);
    pant=0;
}

/* ----- */
/* ----- Array para representar ----- */
/* ----- */

    xs[ta/80]=x_nod[3][0][0];
    resd[ta/80]=resp_des;

/* ----- */
/* ----- Error relativo ----- */
/* ----- */

    error=(resp_des-x_nod[3][0][0]);
    sum_error=sum_error+error;
}
// ***** FIN DEL BUCLE DE ADQUIRIR DATOS *****

    free(data_bin);
    sum_error=0;
}

/* ----- */
/* --- almacenamos los valores de w en un array ----- */
/* ----- */

for (z=1;z<=5;z++)
{
    w_sal[z]=w[2][1][z][0];
}

for (l1=1;l1<=5;l1++)
{
    for (l2=1;l2<=3;l2++)

```

```

    {
        for (l3=1;l3<=10;l3++)
        {
            w_sal[z]=w[l][l1][l2][l3];
            z=z+1;
        }
    }

/*-----*/
/*-----*/
/*----- almacenamos los datos de w en un fichero-----*/
/*-----*/
/*-----*/

ptabla=fopen(fichero,"wb");
fprintf(ptabla,"v");
for (z=1;z<=155;z++)
{
    fprintf(ptabla,"\n%f ",w_sal[z]);
}
fclose(ptabla);

/* ----- */
/* -----Reinicializamos los x_nod de entrada del retardo ----- */
/* ----- */

    for (z=1;z<=5;z++)
    {
        x_nod[2][z][0]=0.5;
    }
    increm=0;

}  /***** FIN BUCLE DE CONVERGENCIA *****/

return true;

}

```

DAT_ASC.cpp

```
/* ----- */
/* Funci3n que pasa datos ascii a binario con un m ximo de 10000 datos */
/* ----- */
#include "stdafx.h"
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>

float (*imp_dat_ascii(char nombre[30]))
{
    char buffer[20];
    float *datos_bin;
    int t=0;

    datos_bin=(float *)calloc(10000,sizeof(float));

    FILE *pf;
    if((pf=fopen(nombre,"r"))==NULL)
    {
        printf("El fichero %s no se puede abrir",nombre);
        exit(1);
    }

    t=0;
    while(fgets(buffer,20,pf)!=NULL)
    {
        if (t<10000)
            datos_bin[t]=atof(buffer); /*conversion de ascii a binario */

        if ((t>=10000))
        {
            printf("Fichero con demasiados datos");
            fclose(pf);
            exit(0);
        }

        t++;
    }
    datos_bin[9999]=t;

    fclose(pf);

    /*      t=numero de muestras del fichero ascii
           datos_bin=fichero ascii de un m ximo de 10000 muestras */

    return (datos_bin);
}
```

FFT.cpp

```
/* ----- */
/*          C lculo de una FFT          */
/* ----- */

#include "stdafx.h"
#include <math.h>

#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <complex>

float (*fft(float *x,int N))
{
    /* variables */

    complex *c;
    complex a2;
    int j,n;
    float a;
    complex sum;
    complex S=(0,0);
    float *c_sal;
    c_sal=(float *)calloc(N,sizeof(float));
    c=(complex *)calloc(240,sizeof(complex));

    /* ----- Funcion FFT ----- */

    for (j=0;j<=N/2;j++)
    {
        for (n=0;n<=N-1;n++)
        {
            a=(2.0*M_PI*j/(N-1))*n;
            a2=complex(0,a);
            sum=x[n]*exp(a2);
            S=S+sum;
        }

        c[j]=S;
        S=(0,0);
        c_sal[j]=log10(abs(c[j]));
    }
    free(c);
    return (c_sal);
}
```

LPC.CPP

```
/* ----- */
/* ----- Funci n LPC ----- */
/* ----- */

#include "stdafx.h"
#include <stdio.h>
#include<stdlib.h>
#include <math.h>

float (*lpc(float X[],int polos)) /*Funcion principal*/
{
    static float a_sal[11]; /*array de salida*/
    float *R; /*array de valores R intermedios*/
    int j=0,m=0;
    float sum=0;
    float G;
    int i=0;
    int z;

    float a[11][11],S=0,*k2;
    float *E;
/* ----- */

    R=(float *)calloc(11,sizeof(float));
    k2=(float *)calloc(11,sizeof(float));
    E=(float *)calloc(11,sizeof(float));

/* ----- */
    for (j=0;j<=polos;j++)
    {
        for (m=0;m<=(239-j);m++)
        {
            sum=X[m]*X[m+j];
            R[j]=R[j]+sum;
        }
    }

/* -- Comenzamos el metodo de Levinson Durbin -- */

    for (j=0;j<=10;j++)
    {
        for (i=0;i<=10;i++)
        {
            a[i][j]=0;
        }
    }
/*-----*/

    E[0]=R[0];
    for (i=1;i<=polos;i++)
    {
        S=0;
        for (j=1;j<=(i-1);j++)
        {
            sum=a[i-1][j]*R[i-j];
            S=S+sum;
        }
        k2[i]=(R[i]-S)/E[i-1];
        a[i][i]=k2[i];
        for (j=1;j<=(i-1);j++)
        {
            a[i][j]=a[i-1][j]-k2[i]*a[i-1][i-j];
        }
        E[i]=(1.0-(k2[i]*k2[i]))*E[i-1];
    }

/*-----c lculo valor G----- */

    S=0.0;
    for (z=1;z<=polos;z++)
```

```

{
    sum=a[polos][z]*R[z];
    S=S+sum;

}
G=sqrt(fabs((double)(R[0]-S)));

/*salida de datos*/
a_sal[0]=G;
for (z=1;z<=polos;z++)
{
    a_sal[z]=a[polos][z];
}

free (R);
free (k2);
free (E);
return(a_sal);
}

```

HAMMING.cpp

```
/* ----- */
/* -- Funcion de ventana Hamming -- */
/* ----- */

#include "stdafx.h"
#include <math.h>

float *hamming(int N)
{
    static float h[241];
    int n;

    for (n=0;n<=N-1;n++)
    {
        h[n]=0.54-0.46*cos((double) (2.0*3.1416*(float)n/(float) (N-1)));
    }

    return (h);
}
```

Verific.dll

Verific2003.cpp

```
/* **** Implementaci3n de una red recurrente de 3 capas **** */
/* **** */
/* **** */
/* Fichero "main" : RED_ANSI.CPP */
/* Funciones a llamar : LPCCEP.CPP */
/* HAMMING.CPP */
/* LPC.CPP */
/* **** */

#include "stdafx.h"
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<conio.h>

/* ----- Declaraci3n de funciones----- */

float (*lpc(float X[],int polos));
float *hamming(int N);

bool _stdcall ResultadoRNN(char* argv1, char* argv2,char* argv3);
bool _stdcall ResultadoRNN(char* argv1, char* argv2,char* argv3)
{
/* ---definimos variables--- */

FILE *pf;
char* fich_w;
char buffer2[20],buffer1[20];
static float wR[4][6][4][11];
static float x_nodR[4][6][11],X2[5];
static float x3[2000];
static float *ham;
int l1,l2,l3,l4,z;
int pant;

fich_w=argv1;
if ((pf=fopen(fich_w,"r"))==NULL)
{
return false;
}

fgets(buffer2,20,pf);
if (buffer2[0]!='V')
{
fclose(pf);
getch();
return false;
}

/* --- colocamos los datos del fichero *.RNN ----- */

for (z=1;z<=5;z++)
{
fgets(buffer2,20,pf);
wR[2][1][z][0]=atof(buffer2);
}

for (l1=1;l1<=5;l1++)
{
for (l2=1;l2<=3;l2++)
{
for (l3=1;l3<=10;l3++)
{
fgets(buffer2,20,pf);
wR[1][l1][l2][l3]=atof(buffer2);
z=z+1;
}
}
}
}
```

```

    }
    }
    fclose (pf);

/* ---- ponemos los valores de 0.5 ----*/

for (z=1;z<=5;z++)
{
    x_nodR[2][z][0]=0.5;
}

/* ----tomamos los datos del fichero *. * -----*/

char* f_voz;
f_voz=argv2;

if ((pf=fopen(f_voz,"rb"))==NULL)
{
    getch();
    fclose(pf);
    return false;
}

static float mayor_C=0;
long cont_C=0;
float valor;
while(!feof(pf))          /* Calculamos el valor mayor del fichero */
{
    fgets(buffer1,20,pf);
    valor=atof(buffer1);
    cont_C=cont_C+1;
    if (fabs(valor)>mayor_C)
    {
        mayor_C=fabs(valor);
    }
}
fclose(pf);

/* llamamos a la función hamming */
ham=hamming(240);

/*----bucle para tomar 240 datos-----*/

fopen(f_voz,"rb");
long buc=0;
static float S1[6];
float S2=0.0,sum;
static float datos1[240], datos2[240];
static float *punt1,*punt2;
fpos_t desp;
desp=(fpos_t)0;

/*-----*/
/* 1 capa de salida */

S2=0.0;
for (z=1;z<=5;z++)
{
    sum=x_nodR[2][z][0]*wR[2][1][z][0];
    S2=S2+sum;
}
x_nodR[3][0][0]=1.0/(1.0+exp((double)-S2)); /* actual salida de red */

/* retardo */

for (z=1;z<=5;z++)
{
    x_nodR[1][1][z]=x_nodR[2][z][0];
}

/*-----*/

```

```

do
{
fsetpos(pf,&desp);
for (z=0;z<=239;z++)
{
fgets(buffer2,20,(pf));
if(z==79)
{
fgetpos(pf,&desp);
}
datos1[z]=(atof(buffer2))*ham[z]/(mayor_C);
}

fsetpos(pf,&desp);

for (z=0;z<=239;z++)
{
fgets(buffer2,20,(pf));
datos2[z]=(atof(buffer2))*ham[z]/(mayor_C);
}
/* --llamamos a la función lpc -- */

punt1=lpc(datos1,10);
punt2=lpc(datos2,10);

/* colocamos los correspondientes valores lpc en sus correspondientes
tramas */

for (l3=1;l3<=10;l3++)
{
x_nodR[1][2][l3]=((*(punt1+l3)));
x_nodR[1][3][l3]=((*(punt2+l3)));
}

/* comenzamos el calculo */

/*-----*/

for (l1=6;l1<=10;l1++)
{
/* nodos vacios */
x_nodR[1][1][l1]=0.0;
}

/* calculo de función sigmoidal */
for (z=0;z<=6;z++)
{
S1[z]=0.0;
}

for (z=1;z<=5;z++) /* capa intermedia */
{
for (l1=1;l1<=3;l1++)
{
for (l2=1;l2<=10;l2++)
{
sum=x_nodR[1][l1][l2]*wR[1][z][l1][l2];
S1[z]=S1[z]+sum;
}
}
x_nodR[2][z][0]=(1.0/(1.0+exp((double)-S1[z]))); /* tomando B=1 */
X2[z-1]=x_nodR[2][z][0];
}

/* calculo de función sigmoidal capa de salida */

S2=0.0;
for (z=1;z<=5;z++)
{
sum=x_nodR[2][z][0]*wR[2][1][z][0];
S2=S2+sum;
}
x_nodR[3][0][0]=1.0/(1.0+exp((double)-S2)); /* actual salida de red */

x3[buc/80]=x_nodR[3][0][0];

```

```

/* -- realizamos el retardo Z-1 --- */
for (z=1; z<=5; z++)
{
    x_nodR[1][1][z]=x_nodR[2][z][0];
}
/* representamos la capa 2 */

    buc=buc+80;
} while(!feof(pf));

fclose(pf);

/* Guardamos los resultados en un fichero */

pf=fopen(argv3,"wb");
for (z=1; z<=buc/80; z++)
{
    fprintf(pf,"%i \r\n", (int) (x3[z]*100));
}
fclose(pf);

    return true;
}

```

Proyecto Visual Basic

frmAudioRecorder.frm

```
Begin VB.Form AudioRecorder
    BorderStyle      = 1  'Fixed Single
    Caption          = "Doctorado UNED"
    ClientHeight     = 10140
    ClientLeft       = 5325
    ClientTop        = 9930
    ClientWidth      = 13035
    Icon             = "frmAudioRecorder.frx":0000
    LinkTopic        = "Form1"
    MaxButton        = 0  'False
    ScaleHeight      = 10140
    ScaleWidth       = 13035
    StartUpPosition = 2  'CenterScreen
    Begin VB.CheckBox chkRCX
        Caption      = "Conectar con RCX"
        Height       = 255
        Left         = 360
        TabIndex     = 22
        Top          = 4320
        Width        = 1935
    End
    Begin VB.PictureBox imgSalida
        BackColor    = &H00FFFFFF&
        Height       = 1575
        Index        = 5
        Left         = 6480
        ScaleHeight  = 101
        ScaleMode    = 3  'Pixel
        ScaleWidth   = 413
        TabIndex     = 21
        Top          = 120
        Width        = 6255
    End
    Begin VB.PictureBox imgSalida
        BackColor    = &H00FFFFFF&
        Height       = 1575
        Index        = 4
        Left         = 6480
        ScaleHeight  = 101
        ScaleMode    = 3  'Pixel
        ScaleWidth   = 413
        TabIndex     = 20
        Top          = 8520
        Width        = 6255
    End
    Begin VB.PictureBox imgSalida
        BackColor    = &H00FFFFFF&
        Height       = 1575
        Index        = 3
        Left         = 6480
        ScaleHeight  = 101
        ScaleMode    = 3  'Pixel
        ScaleWidth   = 413
        TabIndex     = 19
        Top          = 6840
        Width        = 6255
    End
    Begin VB.PictureBox imgSalida
        BackColor    = &H00FFFFFF&
        Height       = 1575
        Index        = 2
        Left         = 6480
        ScaleHeight  = 101
        ScaleMode    = 3  'Pixel
        ScaleWidth   = 413
        TabIndex     = 18
        Top          = 5160
        Width        = 6255
    End
    Begin VB.PictureBox imgSalida
        BackColor    = &H00FFFFFF&
        Height       = 1575
        Index        = 1
        Left         = 6480
        ScaleHeight  = 101
        ScaleMode    = 3  'Pixel
        ScaleWidth   = 413
        TabIndex     = 17
        Top          = 3480
        Width        = 6255
    End
End
```

```

        Index          = 1
        Left           = 6480
        ScaleHeight    = 101
        ScaleMode      = 3 'Pixel
        ScaleWidth     = 413
        TabIndex       = 17
        Top            = 3480
        Width          = 6255
End
Begin VB.PictureBox imgSalida
    BackColor          = &H00FFFFFF&
    Height             = 1575
    Index              = 0
    Left               = 6480
    ScaleHeight        = 101
    ScaleMode          = 3 'Pixel
    ScaleWidth         = 413
    TabIndex           = 16
    Top                = 1800
    Width              = 6255
End
Begin VB.TextBox Text5
    Height             = 285
    Left               = 5640
    TabIndex           = 10
    Top                = 8520
    Width              = 615
End
Begin VB.TextBox Text4
    Height             = 285
    Left               = 5640
    TabIndex           = 9
    Top                = 6840
    Width              = 615
End
Begin VB.TextBox Text3
    Height             = 285
    Left               = 5640
    TabIndex           = 8
    Top                = 5160
    Width              = 615
End
Begin VB.TextBox Text2
    Height             = 285
    Left               = 5640
    TabIndex           = 7
    Top                = 3480
    Width              = 615
End
Begin VB.TextBox Text1
    Height             = 285
    Left               = 5640
    TabIndex           = 6
    Top                = 1800
    Width              = 615
End
Begin VB.CommandButton cmdPlay
    Caption            = "Play"
    Enabled             = 0 'False
    Height             = 495
    Left               = 3720
    TabIndex           = 2
    ToolTipText        = "Play the recording"
    Top                = 480
    Width              = 1215
End
Begin VB.CommandButton cmdStop
    Caption            = "Stop"
    Enabled             = 0 'False
    Height             = 495
    Left               = 1800
    TabIndex           = 1
    ToolTipText        = "Stop recording or playing"
    Top                = 480
    Width              = 1215
End
Begin VB.CommandButton cmdRecord

```

```

Caption      = "Record"
Height       = 495
Left         = 360
TabIndex     = 0
ToolTipText  = "Start recording immediate"
Top          = 480
Width        = 1215
End
Begin VB.Frame Frame5
Caption      = "Control Audio"
Height       = 855
Left         = 240
TabIndex     = 5
Top          = 240
Width        = 4935
End
Begin VB.Timer Timer2
Interval     = 200
Left         = 1200
Top          = 3480
End
Begin VB.Frame Frame4
Height       = 1815
Left         = 240
TabIndex     = 3
Top          = 1320
Width        = 4815
Begin VB.Label StatisticsLabel
BackColor    = &H00000000&
Caption      = " "
ForeColor    = &H0000FF00&
Height       = 1455
Left         = 120
TabIndex     = 4
ToolTipText  = "Information about the recording"
Top          = 240
Width        = 4575
End
End
Begin VB.Frame Frame1
Caption      = "Resultado"
Height       = 3375
Left         = 360
TabIndex     = 23
Top          = 4800
Width        = 4575
Begin VB.Label lblNota
Alignment    = 2 'Center
BeginProperty Font
Name         = "Arial"
Size         = 48
Charset      = 0
Weight       = 400
Underline    = 0 'False
Italic       = 0 'False
Strikethrough = 0 'False
EndProperty
ForeColor    = &H000000FF&
Height       = 1575
Left         = 480
TabIndex     = 24
Top          = 960
Width        = 3495
End
End
Begin VB.Label Label5
Caption      = "Sol"
Height       = 255
Left         = 5280
TabIndex     = 15
Top          = 8520
Width        = 495
End
Begin VB.Label Label4
Caption      = "Fa"
Height       = 255
Left         = 5280

```

```

        TabIndex      = 14
        Top           = 6840
        Width         = 375
    End
    Begin VB.Label Label3
        Caption        = "Mi"
        Height         = 255
        Left           = 5280
        TabIndex       = 13
        Top            = 5160
        Width          = 375
    End
    Begin VB.Label Label2
        Caption        = "Re"
        Height         = 255
        Left           = 5280
        TabIndex       = 12
        Top            = 3480
        Width          = 375
    End
    Begin VB.Label Label1
        Caption        = "Do"
        Height         = 255
        Left           = 5280
        TabIndex       = 11
        Top            = 1800
        Width          = 375
    End
    Begin VB.Menu frmFichero
        Caption        = "Archivo"
        Begin VB.Menu frmSalir
            Caption     = "Salir"
        End
    End
    Begin VB.Menu frmConfig
        Caption        = "Configuración"
        Begin VB.Menu frmAudio
            Caption     = "Audio..."
        End
        Begin VB.Menu frmMenuLego
            Caption     = "Lego..."
        End
    End
    Begin VB.Menu mnuHerramientas
        Caption        = "Herramientas"
        Begin VB.Menu mnuEntrenamiento
            Caption     = "Entrenamiento..."
        End
    End
    Begin VB.Menu mnuAyuda
        Caption        = "Ayuda"
        Begin VB.Menu mnuAcercaDe
            Caption     = "Acerca de..."
        End
    End
End
Attribute VB_Name = "AudioRecorder"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
'DLL en C encargada de implementar la red y obtener la salida
Private Declare Function ResultadoRNN Lib "verific.dll" (ByVal rnn As String, ByVal
DatosVoz As String, ByVal Salida As String) As Boolean
Const AppName = "Doctorado UNED 2003"

Private Sub cmdRecord_Click()
    Dim settings As String
    Dim Alignment As Integer

    Alignment = Channels * Resolution / 8

    settings = "set capture alignment " & CStr(Alignment) & " bitspersample " &
CStr(Resolution) & " samplespersec " & CStr(Rate) & " channels " & CStr(Channels) & "
bytespersec " & CStr(Alignment * Rate)

```

```

WaveReset
WaveSet
WaveRecord
WaveRecordingStartTime = Now
cmdStop.Enabled = True 'Enable el botón de "STOP"
cmdPlay.Enabled = False 'Disable el botón de "PLAY"
cmdRecord.Enabled = False 'Disable el botón de "RECORD"
End Sub

Private Sub cmdStop_Click()
WaveStop

cmdPlay.Enabled = True
cmdStop.Enabled = False

If WaveRecording Then WaveRecordingReady = True
WaveRecordingStopTime = Now
WaveRecording = False
WavePlaying = False
WaveSaveAs (App.Path & "/voz.wav") 'Guardamos el resultado en un fichero .wav
Wav2Txt 'Convertimos el .wav a .txt para pasar el resultado a la red. Es más cómodo
de trabajar

'Pasamos el fichero voz.dat por cada una de las redes entrenadas y obtenemos la
salida
If (ResultadoRNN("rnn/do.rnn", "voz.dat", "resultado/salida1.dat") = False) Then
MsgBox ("Error de lectura de fichero"): End
If (ResultadoRNN("rnn/re.rnn", "voz.dat", "resultado/salida2.dat") = False) Then
MsgBox ("Error de lectura de fichero"): End
If (ResultadoRNN("rnn/mi.rnn", "voz.dat", "resultado/salida3.dat") = False) Then
MsgBox ("Error de lectura de fichero"): End
If (ResultadoRNN("rnn/fa.rnn", "voz.dat", "resultado/salida4.dat") = False) Then
MsgBox ("Error de lectura de fichero"): End
If (ResultadoRNN("rnn/sol.rnn", "voz.dat", "resultado/salida5.dat") = False) Then
MsgBox ("Error de lectura de fichero"): End

RepresentarEntrada 'Función de representación de entrada
CalcularSalida 'Función que calcula la salida optima
cmdRecord.Enabled = True
End Sub

Private Sub cmdPlay_Click()
WavePlayFrom (0)
WavePlaying = True
cmdStop.Enabled = True
cmdPlay.Enabled = False
End Sub

Private Sub RepresentarEntrada()
'Función que representa gráficamente la forma de onda de la entrada
ii = 0
imgSalida(5).Cls
NumFichero = FreeFile
Open App.Path & "/voz.dat" For Input As #NumFichero
Do
ii = ii + 1 / 80
Line Input #NumFichero, strLinea
imgSalida(5).Line -(ii, 50 - CInt(strLinea))
Loop Until EOF(NumFichero)

Close #NumFichero
End Sub

Private Sub CalcularSalida()
Dim ValorSalida(6) As Integer
'Representamos la forma de onda de cada salida
For i = 1 To 5
ii = 0
imgSalida(i - 1).Cls
NumFichero = FreeFile
Open App.Path & "/resultado/salida" & i & ".dat" For Input As #NumFichero
Do
ii = ii + 1
Line Input #NumFichero, strLinea
'Sumamos al ValorSalida, si sobrepasamos el valor de 75 en la salida
If CInt(strLinea) >= 75 Then ValorSalida(i) = ValorSalida(i) + 1
imgSalida(i - 1).Line -(ii, 100 - CInt(strLinea))
Loop Until EOF(NumFichero)

```

```

        Close #NumFichero
    Next i

    Text1.Text = ValorSalida(1)
    Text2.Text = ValorSalida(2)
    Text3.Text = ValorSalida(3)
    Text4.Text = ValorSalida(4)
    Text5.Text = ValorSalida(5)

    'Calculamos el ValorSalida Mayor
    Mayor = ValorSalida(1)
    SalidaMayor = 1
    For i = 2 To 5
        If ValorSalida(i) > Mayor Then
            Mayor = ValorSalida(i)
            SalidaMayor = i
        End If
    Next i
    If Mayor = 0 Then SalidaMayor = 0
    Select Case SalidaMayor
        Case 0
            lblNota.Caption = "?"
        Case 1
            lblNota.Caption = "DO"
        Case 2
            lblNota.Caption = "RE"
        Case 3
            lblNota.Caption = "MI"
        Case 4
            lblNota.Caption = "FA"
        Case 5
            lblNota.Caption = "SOL"
    End Select
    'Salida al lego. Le informamos de la nota que debe tocar
    If chkRCX.Value = 1 And SalidaMayor > 0 Then frmLego.Nota_Click (SalidaMayor
- 1)

End Sub

Private Sub Form_Load()
    WaveReset

    Rate = CLng(GetSetting("AudioRecorder", "Startup", "Rate", "8000"))
    Channels = CInt(GetSetting("AudioRecorder", "Startup", "Channels", "1"))
    Resolution = CInt(GetSetting("AudioRecorder", "Startup", "Resolution", "8"))

    WaveRecordingImmediate = True
    WaveRecordingReady = False
    WaveRecording = False
    WavePlaying = False

    WaveSet

End Sub

Private Sub Form_Unload(Cancel As Integer)
    WaveClose
    Call SaveSetting("AudioRecorder", "Startup", "Rate", CStr(Rate))
    Call SaveSetting("AudioRecorder", "Startup", "Channels", CStr(Channels))
    Call SaveSetting("AudioRecorder", "Startup", "Resolution", CStr(Resolution))
    Call SaveSetting("AudioRecorder", "Startup", "WaveFileName", WaveFileName)
    Call SaveSetting("AudioRecorder", "Startup", "WaveAutomaticSave",
CStr(WaveAutomaticSave))
    If WaveRenameNecessary Then
        Name WaveShortFileName As WaveLongFileName
        WaveRenameNecessary = False
        WaveShortFileName = ""
    End If
End
End Sub

```

```

Private Sub frmAudio_Click()
Dim strWhat As String

    cmdRecord.Enabled = True
    cmdStop.Enabled = False
    cmdPlay.Enabled = False

    frmSettings.Show vbModal
End Sub

Private Sub frmMenuLego_Click()
frmLego.Show vbModal
End Sub

Private Sub frmSalir_Click()
End
End Sub

Private Sub mnuAcercaDe_Click()
frmAbout.Show vbModal
End Sub

Private Sub mnuEntrenamiento_Click()
frmEntrenamiento.Show vbModal
End Sub

Private Sub Timer2_Timer()
    Dim RecordingTimes As String
    Dim msg As String

    RecordingTimes = "Start time: " & WaveRecordingStartTime & vbCrLf _
        & "Stop time: " & WaveRecordingStopTime

    WaveStatistics

    StatisticsLabel.Caption = WaveStatisticsMsg

    WaveStatus
    If WaveStatusMsg <> AudioRecorder.Caption Then AudioRecorder.Caption = WaveStatusMsg
    If InStr(AudioRecorder.Caption, "stopped") > 0 Then
        cmdStop.Enabled = False
        cmdPlay.Enabled = True
    End If

End Sub

Private Sub Wav2Txt()
    Dim FileNumber As Integer
    Dim i As Single
    Dim Temp As Byte

    i = 45 ' Set I To 44, since the wave sample is begin at Byte 44.

    FileNumber = FreeFile
    Open "voz.wav" For Binary As #FileNumber
    Open "voz.dat" For Output As #FileNumber + 1

    Do
        Get #FileNumber, i, Temp
        Num = Temp - (256 / 2)
        If CInt(Num) = 0 Then Num = 0.1
        Print #FileNumber + 1, Num
        i = i + 1

    Loop Until EOF(FileNumber)
    Close #FileNumber
    Close #FileNumber + 1

```

```

End Sub
frmEntrenamiento.frm

VERSION 5.00
Begin VB.Form frmEntrenamiento
    Caption           = "Entrenamiento"
    ClientHeight      = 3165
    ClientLeft        = 4695
    ClientTop         = 3615
    ClientWidth       = 5880
    Icon              = "frmEntrenamiento.frx":0000
    LinkTopic         = "Form1"
    ScaleHeight       = 3165
    ScaleWidth        = 5880
    Begin VB.TextBox txtSalida
        Height         = 375
        Left           = 3240
        TabIndex       = 1
        Text           = "rnn/do.rnn"
        Top            = 1200
        Width          = 1935
    End
    Begin VB.TextBox txtIteraciones
        Alignment       = 1 'Right Justify
        Height          = 285
        Left           = 1440
        TabIndex       = 2
        Text           = "100"
        Top            = 2520
        Width          = 495
    End
    Begin VB.TextBox txtEntrada
        Height         = 375
        Left           = 3240
        TabIndex       = 0
        Text           = "do.txt"
        Top            = 600
        Width          = 1935
    End
    Begin VB.CommandButton btnAceptar
        Caption        = "Aceptar"
        Height         = 495
        Left           = 4200
        TabIndex       = 4
        Top            = 2400
        Width          = 1455
    End
    Begin VB.Frame Frame1
        Caption        = "Ficheros de datos"
        Height         = 2055
        Left           = 120
        TabIndex       = 5
        Top            = 120
        Width          = 5655
        Begin VB.Label Label1
            Caption     = "Fichero de salida con la red entrenada"
            Height      = 495
            Left        = 360
            TabIndex    = 7
            Top         = 1080
            Width       = 2295
        End
        Begin VB.Label Label4
            Caption     = "Fichero de entrada con los datos de los ficheros de voz"
            Height      = 495
            Left        = 360
            TabIndex    = 6
            Top         = 360
            Width       = 2415
        End
    End
    Begin VB.Label Label3
        Caption     = "Iteraciones:"
        Height      = 375
        Left        = 360
        TabIndex    = 3
        Top         = 2520
    End
End

```

```

        Width          = 1095
    End
End
Attribute VB_Name = "frmEntrenamiento"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit
Private Declare Function EntrenamRNN Lib "entrenam.dll" (ByVal FicheroEntrada As String,
ByVal Iteraciones As Integer, ByVal FicheroSalida As String) As Boolean

Private Sub btnAceptar_Click()
Dim boolEntrenam As Boolean
Dim Iteraciones As String
    If txtEntrada.Text = "" Then
        MsgBox ("Por favor, introduce el nombre del fichero de entrada de datos")
        Exit Sub
    End If
    If txtSalida.Text = "" Then
        MsgBox ("Por favor, introduce el nombre del fichero de salida de la red
entrenada")
        Exit Sub
    End If
    Iteraciones = txtIteraciones.Text
    Me.MousePointer = 11
    btnAceptar.Enabled = False
    On Error Resume Next
    boolEntrenam = EntrenamRNN(txtEntrada.Text, Iteraciones, txtSalida.Text)
    Me.MousePointer = 1
    btnAceptar.Enabled = True
End Sub

```

frmLego.frm

VERSION 5.00

Object = "{D6CD40C0-A522-11D0-9800-D3C9B35D2C47}#1.0#0"; "spirit.ocx"

Begin VB.Form frmLego

Caption = "Lego"
ClientHeight = 4755
ClientLeft = 3705
ClientTop = 3615
ClientWidth = 3465
Icon = "frmLego.frx":0000
LinkTopic = "Form1"
ScaleHeight = 4755
ScaleWidth = 3465

Begin VB.TextBox txtPrograma

Height = 285
Left = 1560
TabIndex = 15
Text = "1"
Top = 3360
Width = 615

End

Begin VB.TextBox txtSensorLuz

Height = 285
Left = 1560
TabIndex = 14
Text = "48"
Top = 2640
Width = 615

End

Begin VB.TextBox Text2

Height = 285
Left = 1560
TabIndex = 10
Top = 1440
Width = 615

End

Begin VB.CommandButton Nota

Caption = "sol"
Height = 375
Index = 4
Left = 2640
TabIndex = 8
Top = 240
Width = 495

End

Begin VB.CommandButton Nota

Caption = "fa"
Height = 375
Index = 3
Left = 2040
TabIndex = 7
Top = 240
Width = 495

End

Begin VB.CommandButton Nota

Caption = "mi"
Height = 375
Index = 2
Left = 1440
TabIndex = 6
Top = 240
Width = 495

End

Begin VB.CommandButton Nota

Caption = "re"
Height = 375
Index = 1
Left = 840
TabIndex = 5
Top = 240
Width = 495

End

Begin VB.TextBox Text1

Height = 285
Left = 1560

```

        TabIndex      = 4
        Top           = 1050
        Width         = 615
End
Begin VB.CommandButton PoolVariables
    Caption          = "Obtener datos"
    Height           = 375
    Left             = 1320
    TabIndex         = 3
    Top              = 1920
    Width            = 1455
End
Begin VB.CommandButton Nota
    Caption          = "do"
    Height           = 375
    Index            = 0
    Left             = 240
    TabIndex         = 2
    Top              = 240
    Width            = 495
End
Begin VB.CommandButton ProgramaRCX
    Caption          = "Programa al RCX"
    Height           = 615
    Left             = 360
    TabIndex         = 1
    Top              = 3840
    Width            = 2415
End
Begin SPIRITLib.Spirit PBrickCtrl
    Height           = 495
    Left             = 2880
    TabIndex         = 0
    Top              = 3960
    Visible          = 0 'False
    Width            = 495
    _Version         = 65536
    _ExtentX         = 873
    _ExtentY         = 873
    _StockProps      = 0
End
Begin VB.Frame Frame1
    Caption          = "Notas"
    Height           = 735
    Left             = 120
    TabIndex         = 12
    Top              = 0
    Width            = 3135
End
Begin VB.Label Label4
    Caption          = "Programa:"
    Height           = 255
    Left             = 360
    TabIndex         = 16
    Top              = 3360
    Width            = 975
End
Begin VB.Label Label1
    Caption          = "Sensor Luz:"
    Height           = 375
    Left             = 360
    TabIndex         = 13
    Top              = 2640
    Width            = 1095
End
Begin VB.Line Line3
    X1               = 0
    X2               = 3480
    Y1               = 3120
    Y2               = 3120
End
Begin VB.Line Line2
    X1               = 0
    X2               = 3480
    Y1               = 2400
    Y2               = 2400
End
End

```

```

Begin VB.Line Line1
    X1          = 0
    X2          = 3480
    Y1          = 840
    Y2          = 840
End
Begin VB.Label Label3
    Caption      = "Nota Anterior:"
    Height       = 375
    Left         = 360
    TabIndex     = 11
    Top          = 1440
    Width        = 1095
End
Begin VB.Label Label2
    Caption      = "Nota Actual:"
    Height       = 255
    Left         = 360
    TabIndex     = 9
    Top          = 1080
    Width        = 1095
End
End
Attribute VB_Name = "frmLego"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
'Option explicit

Private Sub Form_Load()
    PBrickCtrl.InitComm
End Sub

Public Sub Nota_Click(Index As Integer)
    PBrickCtrl.SetVar 2, CTE, PBrickCtrl.Poll(VAR, 1) 'en la variable 2 el valor
anterior
    PBrickCtrl.SetVar 1, CTE, (1 * (Index + 1))      'en la variable 1 el valor actual
    PBrickCtrl.SetVar 3, CTE, PBrickCtrl.Poll(VAR, 1) - PBrickCtrl.Poll(VAR, 2) 'en la
variable 1 el valor actual
End Sub

Private Sub PoolVariables_Click()

ValorActual = PBrickCtrl.Poll(VAR, 1)
ValorAnterior = PBrickCtrl.Poll(VAR, 2)
    Select Case ValorActual
        Case 1
            Text1.Text = "do"
        Case 2
            Text1.Text = "re"
        Case 3
            Text1.Text = "mi"
        Case 4
            Text1.Text = "fa"
        Case 5
            Text1.Text = "sol"
    End Select
    Select Case ValorAnterior
        Case 1
            Text2.Text = "do"
        Case 2
            Text2.Text = "re"
        Case 3
            Text2.Text = "mi"
        Case 4
            Text2.Text = "fa"
        Case 5
            Text2.Text = "sol"
    End Select
End Sub

Private Sub ProgramaRCX_Click()
    With PBrickCtrl
        .SelectPrgm txtPrograma.Text - 1      'seleccionamos el programa
        .BeginOfTask PRINCIPAL                'comienza la tarea principal
    End With
End Sub

```

```

'Configuramos los sensores y motores
.SetSensorType SENSOR_1, TIPO_SWITCH
.SetSensorType SENSOR_3, TIPO_LUZ
.SetSensorMode SENSOR_3, PORCENT_MODO, 0
.SetPower MOTOR_A + MOTOR_C, CTE, 6

'*****
'Bucle principal infinito
'*****
.Loop CTE, 0

    '*****
    'La variable 3 rige el comportamiento del robot
    'Los valores posibles son
    '100    robot parado
    '0      robot toca nota
    '1      robot adelanta 1 nota
    '2      robot adelanta 2 notas
    '.....
    '.....
    '-1     robot retrasa 1 nota
    '-2     robot retrasa 1 nota
    '.....
    '.....
    '*****

    '*****
    'Movimiento del robot para situarse en la nota correspondiente
    '*****
    .If VAR, 3, DIF, CTE, 100      'si var 3 no es 100
        .If VAR, 3, MENQ, CTE, 0  'si variable 3 es menor que 0
            .SetRwd MOTOR_C      'motor atrás
        .EndIf
        .If VAR, 3, MAYQ, CTE, 0  'si variable 3 es mayor que 0
            .SetFwd MOTOR_C      'motor adelante
        .EndIf
        .On MOTOR_C              'motor C on
        .AbsVar 3, VAR, 3        'quitamos signo a la variable 3
        .Loop VAR, 3             'loop en función de var 3
        .Wait 2, 20              'esperamos 200ms, en ese tiempo no puede
haber 2 pasos de nota

        .While SENVAL, SENSOR_3, MAYQ, CTE,
txtSensorLuz.Text 'No hacemos nada mientras que el sensor
        .EndWhile
'de luz sea mayor que el umbral
        .EndLoop
        .SetVar 3, CTE, 0        'Mandamos al robot a tocar nota poniendo
var 3 igual a 0
        .EndIf

    '*****
    'Movimiento del brazo del robot para tocar nota
    '*****
    .If VAR, 3, IG, CTE, 0        'si var 3 es 0
        .Off MOTOR_C             'paramos el motor de movimiento
        .SetFwd MOTOR_A          'encendemos el motor del brazo
        .While SENVAL, SENSOR_1, IG, CTE, 0 'mientras no toquemos el
sensor de colision
        .EndWhile
        .SetRwd MOTOR_A          'cambiamos el sentido de giro del motor
        .Wait CTE, 50            'dejamos medio segundo que suba el brazo
        .Off MOTOR_A            'paramos el motor del brazo
        .SetVar 3, CTE, 100      'variable 3 lista para recibir nueva
nota
        .EndIf
    .EndLoop
    .EndOfTask
End With
End Sub

```

frmSettings.frm

VERSION 5.00

```
Begin VB.Form frmSettings
    Caption           = "Configuración"
    ClientHeight      = 2325
    ClientLeft        = 5355
    ClientTop         = 4755
    ClientWidth       = 5775
    Icon              = "frmSettings.frx":0000
    LinkTopic         = "Form1"
    MinButton         = 0   'False
    ScaleHeight       = 2325
    ScaleWidth        = 5775
    Begin VB.CommandButton cmdOke
        Caption       = "Ok"
        Height        = 375
        Left          = 2040
        TabIndex      = 12
        Top           = 1680
        Width         = 1575
    End
Begin VB.Frame Frame3
    Caption           = "Resolution"
    Height            = 1095
    Left              = 3960
    TabIndex          = 9
    Top               = 120
    Width             = 1575
    Begin VB.OptionButton opt8bits
        Caption       = "8 bits"
        Height        = 255
        Left          = 360
        TabIndex      = 11
        Top           = 360
        Width         = 855
    End
    Begin VB.OptionButton opt16bits
        Caption       = "16 bits"
        Height        = 255
        Left          = 360
        TabIndex      = 10
        Top           = 720
        Width         = 855
    End
End
Begin VB.Frame Frame2
    Caption           = "Canales"
    Height            = 1095
    Left              = 2040
    TabIndex          = 6
    Top               = 120
    Width             = 1575
    Begin VB.OptionButton optMono
        Caption       = "mono"
        Height        = 255
        Left          = 360
        TabIndex      = 8
        Top           = 720
        Width         = 855
    End
    Begin VB.OptionButton optStereo
        Caption       = "stereo"
        Height        = 255
        Left          = 360
        TabIndex      = 7
        Top           = 360
        Width         = 855
    End
End
Begin VB.Frame Frame1
    Caption           = "Frec muestreo (Hz)"
    Height            = 2055
    Left              = 120
    TabIndex          = 0
    Top               = 120
```

```

Width          = 1575
Begin VB.OptionButton optRate6000
    Caption      = "6000"
    Height       = 315
    Left         = 360
    TabIndex     = 5
    Top          = 1560
    Width        = 1095
End
Begin VB.OptionButton optRate8000
    Caption      = "8000"
    Height       = 315
    Left         = 360
    TabIndex     = 4
    Top          = 1260
    Width        = 1095
End
Begin VB.OptionButton optRate11025
    Caption      = "11025"
    Height       = 315
    Left         = 360
    TabIndex     = 3
    Top          = 960
    Width        = 1095
End
Begin VB.OptionButton optRate22050
    Caption      = "22050"
    Height       = 315
    Left         = 360
    TabIndex     = 2
    Top          = 660
    Width        = 1095
End
Begin VB.OptionButton optRate44100
    Caption      = "44100"
    Height       = 315
    Left         = 360
    TabIndex     = 1
    Top          = 360
    Width        = 1095
End
End
Attribute VB_Name = "frmSettings"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit

Private Sub cmdOke_Click()
    Unload Me
End Sub

Private Sub Form_Load()
    Select Case Rate
    Case 44100
        optRate44100.Value = True
    Case 22050
        optRate22050.Value = True
    Case 11025
        optRate11025.Value = True
    Case 8000
        optRate8000.Value = True
    Case 6000
        optRate6000.Value = True
    End Select

    Select Case Channels
    Case 1
        optMono.Value = True
    Case 2

```

```

        optStereo.Value = True
    End Select

    Select Case Resolution
    Case 8
        opt8bits.Value = True
    Case 16
        opt16bits.Value = True
    End Select

End Sub

Private Sub optRate11025_Click()
    Rate = 11025
    optRate11025.Value = True
End Sub

Private Sub optRate44100_Click()
    Rate = 44100
    optRate44100.Value = True
End Sub

Private Sub optRate22050_Click()
    Rate = 22050
    optRate22050.Value = True
End Sub

Private Sub optRate8000_Click()
    Rate = 8000
    optRate8000.Value = True
End Sub

Private Sub optRate6000_Click()
    Rate = 6000
    optRate6000.Value = True
End Sub

Private Sub optMono_Click()
    Channels = 1
    optMono.Value = True
End Sub

Private Sub optStereo_Click()
    Channels = 2
    optStereo.Value = True
End Sub

Private Sub opt8bits_Click()
    Resolution = 8
    opt8bits.Value = True
End Sub

Private Sub opt16bits_Click()
    Resolution = 16
    opt16bits.Value = True
End Sub

```

RCXDatos.bas

```
Attribute VB_Name = "modLego"
'=====
' Programación de robots Lego MindStorms. Visual Basic.
' Modulo global
' 1.0 versión
'-----
' Declaración de nombres globales de constantes para el RCX
'=====
Option Explicit
'=====
' Introduce aquí o al inicio del programa tus propias constantes
'=====

'-----
' Slots de programas 0 - 4
'=====
Public Const SLOT_1 = 0
Public Const SLOT_2 = 1
Public Const SLOT_3 = 2
Public Const SLOT_4 = 3
Public Const SLOT_5 = 4
'=====
' Nombres de tareas - Modificar los nombres de tareas 1 - 9 por nombres
' más significativos. Para ello no es necesario que hagas las
' sustituciones aquí, puedes hacerlo al principio del programa
'=====
Public Const PRINCIPAL = 0
Public Const TAREA_1 = 1
Public Const TAREA_2 = 2
Public Const TAREA_3 = 3
Public Const TAREA_4 = 4
Public Const TAREA_5 = 5
Public Const TAREA_6 = 6
Public Const TAREA_7 = 7
Public Const TAREA_8 = 8
Public Const TAREA_9 = 9
'=====
' Sonidos de sistema
'=====
Public Const CLICK_SOUND = 0
Public Const BEEP_SOUND = 1
Public Const SWEEP_DOWN_SOUND = 2
Public Const SWEEP_UP_SOUND = 3
Public Const ERRORE_SOUND = 4
Public Const SWEEP_FAST_SOUND = 5

'-----
' Nombres de origen de datos
'=====
Public Const VAR = 0
Public Const TEMPOR = 1
Public Const CTE = 2
Public Const MOTSTA = 3
Public Const RAN = 4
Public Const TACC = 5
Public Const TACS = 6
Public Const MOTCUR = 7
Public Const KEYS = 8
Public Const SENVAL = 9
Public Const SENTIPO = 10
Public Const SENMOD = 11
Public Const SENRAW = 12
Public Const BOOL = 13
Public Const RELOJ = 14
Public Const PBMESS = 15
'=====
' Nombres de sensores
'=====
Public Const SENSOR_1 = 0
Public Const SENSOR_2 = 1
Public Const SENSOR_3 = 2
```

```

'=====
' Nombres de temporizadores
'=====
Public Const TEMPOR_1 = 0
Public Const TEMPOR_2 = 1
Public Const TEMPOR_3 = 2
Public Const TEMPOR_4 = 3
'=====
' Nombres de tacómetro (sólo CyberMaster)
'=====
Public Const TACHO_IZQ = 0
Public Const TACHO_DCHO = 1
'=====
' Nombres de alcance
'=====
Public Const ALCANCE_CORTO = 0
Public Const ALCANCE_LARGO = 1
'=====
' Tipos de sensor
'=====
Public Const NO_TIPO = 0
Public Const TIPO_SWITCH = 1
Public Const TIPO_TEMP = 2
Public Const TIPO_LUZ = 3
Public Const TIPO_ANGULO = 4
'=====
' Modos de sensor
'=====
Public Const RAW_MODAL = 0
Public Const BOOL_MODAL = 1
Public Const TRANS_MODAL_MODAL = 2
Public Const PERIOD_MODAL_MODAL = 3
Public Const PORCENT_MODAL_MODAL = 4
Public Const CELSIUS_MODAL_MODAL = 5
Public Const FAHRENHEIT_MODAL_MODAL = 6
Public Const ANGULO_MODAL_MODAL = 7
'=====
' Nombres de motor (cadenas)
'=====
Public Const MOTOR_A = "0"
Public Const MOTOR_B = "1"
Public Const MOTOR_C = "2"
'=====
' Nombres de salida
'=====
Public Const SALIDA_A = "0"
Public Const SALIDA_B = "1"
Public Const SALIDA_C = "2"
'=====
' Operadores lógicos de comparación
'=====
Public Const MAYQ = 0 'Mayor que
Public Const MENQ = 1 'Menor que
Public Const IG = 2 'Igual
Public Const DIF = 3 'No igual a
'=====
' Miscellaneous
'=====
Public Const SIEMPRE = 0
'=====
' Constantes de tiempo (MS = milisegundo)
'=====
Public Const MS_10 = 1 ' 10 ms
Public Const MS_20 = (2 * MS_10) ' 20 ms
Public Const MS_30 = (3 * MS_10) ' 30 ms
Public Const MS_40 = (4 * MS_10) ' 40 ms
Public Const MS_50 = (5 * MS_10) ' 50 ms
Public Const MS_60 = (6 * MS_10) ' 60 ms
Public Const MS_70 = (7 * MS_10) ' 70 ms
Public Const MS_80 = (8 * MS_10) ' 80 ms
Public Const MS_90 = (9 * MS_10) ' 90 ms
Public Const MS_100 = (10 * MS_10) '100 ms
Public Const MS_200 = (20 * MS_10) '200 ms
Public Const MS_300 = (30 * MS_10) '300 ms
Public Const MS_400 = (40 * MS_10) '400 ms
Public Const MS_500 = (50 * MS_10) '500 ms
Public Const MS_700 = (70 * MS_10) '700 ms

```

```
Public Const SEG_1 = (100 * MS_10)      ' 1 s
Public Const SEG_2 = (2 * SEG_1)        ' 2 s
Public Const SEG_3 = (3 * SEG_1)        ' 3 s
Public Const SEG_5 = (5 * SEG_1)        ' 5 s
Public Const SEG_10 = (10 * SEG_1)       '10 s
Public Const SEG_15 = (15 * SEG_1)      '15 s
Public Const SEG_20 = (20 * SEG_1)      '20 s
Public Const SEG_30 = (30 * SEG_1)      '30 s
Public Const MIN_1 = (60 * SEG_1)       ' 1 mn
```